Dittrich

ATARI® ST

Peeks & Pokes

EIN DATA BECKER BUCH

Dittrich

ATARI® ST

Peeks & Pokes

EIN DATA BECKER BUCH



ISBN 3-89011-148-3

2. überarbeitete Auflage

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Vorwort

Mit dem ATARI ST ist das vielseitige Angebot an Personal-Computern um ein Gerät reicher, das neue Dimensionen bietet. Vielseitigkeit, Bedienungskomfort und vor allem Preis-Leistungsverhältnis dieses Rechners stellen viele andere Computer derselben Klasse in den Schatten. Weitere Pluspunkte sind: die übersichtliche Tastatur, die vielen Schnittstellen, die Speicherkapazität von 512 KByte (und mehr) und das (bald) eingebaute Betriebssystem GEMDOS, welches in Verbindung mit der Maus auch für jeden Laien leicht zu erlernen und zu bedienen ist.

Aber wie sind die vielfältigen Fähigkeiten dieses Gerätes auch für eigene Ideen anzuwenden und auszunutzen? Das mitgelieferte Handbuch beschreibt zwar die Bedienung der (meisten) eingebauten Funktionen, doch bleiben immer noch viele Fragen offen.

Hier soll dieses Buch ansetzen. Es gibt einen Einblick in das Innenleben Ihres Rechners, stellt sozusagen einen "Reiseführer durch die Wunderwelt des ATARI ST" dar. Auf dieser Reise gibt es viele Sehenswürdigkeiten und zahlreiche Hilfen zur besseren Orientierung. Anhand von Programmbeispielen sollen die Fähigkeiten des Computers enthüllt werden; Ihnen, dem "User", wird so die Möglichkeit gegeben, Ihre eigenen Vorstellungen bezüglich der Anwendung des Gerätes zu verwirklichen.

Um solche Variationen an einem bestehenden Gerät vornehmen zu können, fehlt aber so mancher spezielle Befehl, der dies bewerkstelligt. Hier sind wir schon beim Titel dieses Buches. PEEK und POKE sind Befehle der Programmiersprache BASIC, die einen direkten Eingriff in die internen Funktionen des Geräts ermöglichen. Sie können dann mit Hilfe einiger Tabellen, die wichtige Systemparameter enthalten, Manipulationen am mächtigen Betriebssystem des ATARI ST vorzunehmen.

Weiterhin werden auch die Möglichkeiten der Kommunikation mit der Außenwelt betrachtet. Datenaustausch mit anderen Computern, Anschluß und Bedienung eines Telefonmodems werden ebenso behandelt wie Steuerung externer Geräte. Auch der Anschluß anderer Diskettenlaufwerke sowie eines Druckers wird beschrieben.

Bevor Sie nun die Anwendungsbeispiele in diesem Buch ausprobieren, möchte ich noch auf einen Punkt hinweisen: Das vorliegende Gerät ist eines der "ersten Stunde". Sein Betriebssystem befindet sich daher noch nicht bei allen Rechnern fest im eingebauten Speicher, sondern wird im allgemeinen von der Diskette geladen. Dafür existieren bereits zwei verschiedene TOS-Versionen, für die Sie im Anhang eine Tabelle der wichtigsten Adressen finden können. Die im Buch verwendeten Adressen gelten für die 197 KByte-Version. Auch die BASIC-Version, mit der die Beispielprogramme entwickelt wurden, ist eine der ersten funktionierenden Versionen. Die endgültigen BASIC-Interpreter werden wahrscheinlich kürzer sein. Selbstverständlich werden bei größeren Neuerungen, die ATARI auf den Markt bringen wird, diese in weiteren Auflagen einbezogen. Somit steht der Anwendung dieses Buches bei der ganzen ST-Serie nichts im Wege!

Aber nun genug der Einleitung, schließlich wollen Sie ja etwas über Ihren ATARI ST lesen. Mir bleibt dann nur noch, Ihnen viel Vergnügen bei der Verwirklichung Ihrer Ideen mit Hilfe dieses Buches zu wünschen.

Abschließend möchte ich noch meinen Kommilitonen und Freunden für ihre Mitarbeit an der Erstellung dieses Buches danken, durch die es die Fülle an Informationen und Tips erhalten konnte, die Ihnen hiermit vorliegt.

Stefan Dittrich
Gummersbach, März 1986

Inhaltsverzeichnis

1. Ein-Blick in den ATARI ST

Ein neuer Computer übt immer einen ungeheuren Reiz aus. Man ist versucht, sich einfach dranzusetzen und draufloszuprogrammieren. Aber dabei ist es leider oft so, daß man die Besonderheiten des Rechners außer acht läßt und ihn wie einen ganz normalen anderen Computer behandelt.

Doch gerade dies ist beim ATARI ST ausgesprochen schade. Kaum ein anderer Personal-Computer bietet so viele Besonderheiten, die selbst den erfahren Programmierer immer wieder vom Hocker reißen. Das macht nicht nur das sehr starke und zukunftweisende GEM aus, dessen komfortable Bedienung den Rechner geradezu sympathisch macht. Auch die technische Konzeption ist so ausgefeilt, daß es mit ihrer Hilfe möglich ist, den ATARI ST für nahezu jeden Zweck einzusetzen. Um das überschauen zu können, muß man allerdings einen Blick hinter die Kulissen des ST werfen und sich die einzelnen Komponenten ansehen, die ja den ATARI ST eigentlich erst ausmachen. Riskieren wir also mal einen Blick in das Innenleben des Gerätes.

1.1 Innere Konfiguration

Der erste Blick auf den neuen ATARI ST läßt wenig ahnen, was sein Äußeres verbirgt. Oft hört man dann den Ausruf: "Ist ja wirklich ein gutaussehendes Gerät, aber wo ist denn die ganze Technik? Da kann ja nicht viel drin sein!"

Irrtum! In dem relativ unscheinbaren Gehäuse des ST (verglichen z.B. mit einem IBM-PC) verbirgt sich eine Riesenmenge an Technik. Wenn man dann in sein Innenleben blickt, ist man überrascht, wie wenig es enthält und wie viel der Rechner doch leistet. Das liegt daran, daß in dem ATARI ST der neueste Stand der Technik verwendet wurde. Sehen wir uns mal an, was er kann und wie, bzw. womit er das macht.

Zuerst wäre da der Prozessor selbst, der eigentlich die Intelligenz eines Computers ausmacht. Es handelt sich hier um den MC 68000 von Motorola, das größte Bauteil auf der Hauptplatine. Dieser Prozessor hat phantastische Leistungsmerkmale:

* 17 32-Bit Daten- und Adressregister
* Adressierungsbereich von 16 Megabyte
* 56 Maschinenbefehle
* 14 Adressierungsarten, und vieles mehr.

Er arbeitet intern mit 32-Bit-Worten, d.h. mit Werten von bis zu 232 = 4.294.967.296. Herausgeführt sind 16 Datenleitungen, weshalb er auch als 16-Bit-Prozessor bezeichnet wird. Diese großen Zahlen bedeuten einen großen Vorteil, nämlich Geschwindigkeit. Der MC 68000 kann mit wenig Arbeit viele Daten (z.B. Text) verarbeiten. Dazu kommt noch, daß er einen Arbeitstakt von 8 MHz hat. Er führt also bis zu acht Millionen Befehle in der Sekunde aus. Dies ist allerdings ein rein theoretischer Wert, da viele Befehle mehrere Takte benötigen. Dennoch ist die effektive Geschwindigkeit enorm, was man gerade beim ATARI ST z.B. bei der Grafikwiedergabe sieht.

Der Prozessor ist also das Gehirn des Rechners. Die Arbeiter im ST sind zusätzliche Bausteine, die den "Boß" unterstützen. Es ist sogar ein weiterer intelligenter Chip im ST eingebaut, und zwar in der Tastatur. Dieser nennt sich HD 6301V1 und hat sehr wichtige Aufgaben: die Abfrage und Auswertung aller Tastaturbedienungen einschließlich der Funktionstasten F1 bis F10, die Beobachtung der Maus und das Auswerten ihrer Bewegungen sowie die Verwaltung der Joystickaktionen. Außerdem enthält der 6301 eine Echtzeituhr, die in Sekunden zählt. Die Möglichkeiten, die diese Tastatur dem Programmierer eröffnet, sind so vielfältig, daß sogar ein eigenes Kapitel in diesem Buch darüber berichtet, siehe Abschnitt 1.3, "Die intelligente Tastatur".

Doch nun zurück zum Hauptrechner. Hier gibt es noch einige Bausteine, die den ST erst zu dem machen, was er ist. Die hier verwendeten integrierten Schaltkreise, üblicherweise als IC's (Integrated Circuits) bezeichnet, sind speziell für den ATARI 520ST entwickelt. Das Zusammenspiel dieser ICs verwaltet der

Glue (engl. für Kleber). Ein weiterer Baustein ist die MMU, die Memory-Management-Unit. Diese Einheit verwaltet den Arbeitsspeicher, also die 512 KByte. Ausgelegt ist sie sogar für 4 Megabyte, wovon 1 MByte in der neueren Versionen des ST enthalten ist.

Als nächstes ist der Shifter zu erwähnen, der für den Aufbau des Bildes auf dem Monitor sorgt. Er interpretiert auch den Bildschirmspeicher, unabhängig ob Farbbild oder nicht, und gibt das entsprechende Video-Signal an den Monitor aus. Der Monitoranschluß des ATARI liegt also am Shifter. Hier liegen entweder ein Video-Signal am Monochrome-Ausgang oder RGB-Signale an den Rot-, Grün- und Gelb-Ausgängen an. Ein gleichzeitiges Betreiben eines Farb- und eines S/W-Monitores ist somit nicht möglich! Die Information über die Art des Monitors erhält der Rechner durch den Monochrome-Detect Eingang, der entweder auf +5V (Monochrom) oder 0V (RGB) liegt.

Ein weiterer wichtiger Baustein ist der DMA-Controller (DMA = Direct Memory Access). Er kann direkt auf den Speicher zugreifen und dient der Datenübertragung per Diskettenlaufwerk oder Festplatte. Die hohen Geschwindigkeiten, mit denen diese Datenübertragungen stattfinden, würden die CPU (Central Prozessor Unit), d.h. den 68000er überfordern - also macht der DMA-Controller diesen Job.

Die Steuerung der Diskettenstation(en) ist Aufgabe des Floppy-Disk-Controllers WD 1772, der nach Bedarf Daten von der Diskette holt, den Motor startet, den Schreib-/Lesekopf bewegt und die durchgehenden Daten aufbereitet. Der Controller ist in der Lage sowohl ein- wie auch zweiseitige Diskettenlaufwerke anzusteuern. So ist es also kein Problem, zweiseitige Laufwerke an den ATARI ST anzuschließen, wobei das Format (3½ oder 5¼ Zoll) ebenfalls keine Rolle spielt. Die Programmierung des WD 1772 ist recht einfach und wird so gut vom Betriebssystem unterstützt, daß man sich darüber keine Gedanken machen müßte.

Ein weiteres, sehr wichtiges IC ist der Multifunktionschip MFP 68901. Seine Spezialität sind Ein- und Ausgaben. So steuert er den Druckerausgang und die serielle Schnittstelle und ist intern auch für Timerfunktionen zuständig.

Als nächstes wären die zwei ACIAs (Asynchronus Communication Interface Adapter) zu erwähnen, die zur Abwicklung der seriellen Datenübertragung dienen. Der eine ACIA wickelt die Kommunikation mit der Tastatur ab, während das andere für die MIDI-Schnittstelle zuständig ist. Letzteres ist interessant für die Verwendung als Netzwerk-Anschluß, denn diese Bausteine sind programmierbar, wie die anderen Sonder-ICs auch. Schließlich ist eine serielle Schnittstelle, die zudem noch recht schnell arbeitet (im MIDI-Betrieb immerhin 31250 Baud), für viele andere Anwendungen zu gebrauchen.

Und nun noch das Sound-IC YM 2149 des ATARI ST. Dieses IC, hergestellt von der als Orgel-/Synthesizerhersteller bekannten Firma Yamaha, besitzt drei unabhängige Tongeneratoren und einen Rauschgenerator. Die Hüllkurven, Lautstärken, Mischverhältnisse und Tonhöhen sind programmäßig einstellbar. Die Breite des Frequenzspektrums übersteigt bei weitem den hörbaren Bereich. Es reicht vom satten Baß bis (fast) zum Ultraschall.

All diese fähigen Heinzelmännchen sind natürlich programmierbar. Das Sound-Chip einfach mit den BASIC-Befehlen SOUND und WAVE; bei den anderen wird es schon etwas bis sehr kompliziert. Die einfachste Manipulationsmethode ist die Änderung der jeweiligen Parameter. Jedes IC hat einen reservierten Speicherbereich, in den es seine Variablen ablegt. Dort findet dann der Informationsabtausch statt (siehe Abschnitt 2.1). Doch bleiben wir erst einmal auf der Ebene der sicht- und greifbaren Technik.

1.2 Schnittstellen

Die Anwendungsvielfalt eines Rechners steht und fällt mit seinen Möglichkeiten, mit der Außenwelt zu kommunizieren. Das geschieht mittels Schnittstellen, die wir nun betrachten wollen.

Die Datenübertragung von einem Sender zum Empfänger kann auf zweierlei Arten geschehen:

1. Parallel
oder 2. Seriell.

1.2.1 Parallel-Schnittstelle

Parallele Datenübertragung bedeutet, daß dem Empfänger das gesamte Datenwort auf einmal angeboten wird. Der Vorteil dieses Verfahrens liegt in der hohen Geschwindigkeit, die damit erreicht werden kann. Der Nachteil ist allerdings die pro Datenbit benötigte Leitung.

Ein typisches Beispiel einer Parallel-Schnittstelle ist der Centronics-Port. Er wird in der Regel zum Ansteuern eines Druckers verwendet, da er nur in einer Richtung Daten übertragen kann, und zwar vom Computer zum Drucker. Der Centronics-Stecker befindet sich an der Rückseite des ST und trägt die Bezeichnung *Printer.*

Die Verbindung zwischen diesem Stecker und dem Drucker (z.B. EPSON) geschieht über ein mindestens elfpoliges Kabel, und zwar folgendermaßen:

Computer Pin	Leitung		Drucker Pin
1	Strobe	--->	1
2	Data 0	--->	2
3	Data 1	--->	3
4-9	Data 2-7	--->	4-9
11	Busy	<---	11
18	Ground		19

Wie man sieht, führt das BUSY-Signal auf den Computer zurück. Diese Rückleitung wird verwendet als sogenannte Handshake-Leitung und ist für die Sicherheit der Datenübertragung notwendig.

Will der Computer nun dem Drucker ein Zeichen übergeben, so legt er den ASCII-Wert des Zeichens (0-255) binär an die Data-Ausgänge an und teilt dem Drucker über ein LOW-Signal an Strobe mit, daß er dieses Datenwort übernehmen kann. Der Drucker legt nun dieses Byte in seinem Speicher ab. Wenn er weitere Daten übernehmen kann, teilt er dies durch ein LOW-Signal an BUSY dem Computer mit. Dieser muß also so lange mit der Datenübertragung warten, bis das BUSY-Signal wieder auf LOW fällt.

Wenn nun der Drucker nicht eingeschaltet oder angeschlossen ist, so bekommt der Computer keine Antwort über die BUSY-Leitung. Er wartet dann noch eine Weile, bis er dann schließlich aufgibt und gegebenfalls eine Fehlermeldung ausgibt. Diese "Auszeit", das Timeout, ist beim ATARI ST mit 30 Sekunden recht hoch. Das liegt daran, daß das sogenannte ACK-Signal (acknowledge = Erkennung) nicht angeschlossen ist, wodurch der Rechner sofort erkennen könnte, ob der Drucker bereit ist. Der ATARI ST muß also warten, ob ein Busy-Signal kommt. Da der Drucker ja gerade beschäftigt sein könnte, muß ihm ausreichend Zeit gegeben werden, seine augenblickliche Tätigkeit zu Ende zu führen.

Eine Besonderheit bietet der ATARI ST bei dieser Schnittstelle, die eigentlich völlig untypisch ist. Er kann nämlich vom Parallelport lesen. Diese Anwendungsmöglichkeit ist enorm vielfältig; besitzt man damit doch die Möglichkeit, digitale Schaltungen anzuschließen und deren gelieferte Signale auszuwerten. Die Abfrage dieser Daten aus dem Druckeranschluß geschieht mittels des INP(0)-Befehls der Programmiersprache BASIC, der das anliegende Datenbyte ausgibt. Somit kann der ST auch Daten aufnehmen, die eigentlich für einen Drucker bestimmt sind. Man braucht hierfür nur den Centronics-Ausgang eines Computers mit dem Parallel-Stecker des ATARI verbinden, wobei die An-

schlüsse 1 und 11 überkreuzt angeschlossen werden. Danach startet man dann im ST folgendes kleine BASIC-Programm:

```
10   rem *** Drucker - Simulation ***
20   x = inp(0)            : rem Datum einlesen
30   print chr$(x);        : rem Zeichen ausgeben
40   goto 20               : rem Endlos-Schleife
```

Alle Druckerausgaben, die der andere Rechner nun ausgibt, werden auf dem OUTPUT-Fenster des ST augegeben. Auf diese Weise kann auch eine Datenübertragung mit Computern ohne RS-232-Ausgang erfolgen.

Aber abgesehen von dieser interessanten Verwendungsmöglichkeit des Parallelports kann noch eine Steuerung durch den ST vorgenommen werden. Ein schönes Beispiel dafür sind die Bausätze, die von Fischer-Technik für Computer angeboten werden. Mit diesen Bauteilen lassen sich kleine Roboterarme oder einfache Plotter aufbauen, die vom Rechner gesteuert werden können. Aber auch eine Aufzugssteuerung ist möglich, für die ja der Parallelport des ST hervorragend geeignet ist. Für eine solche Aufgabe muß der Rechner Daten aufnehmen, auswerten und Steuerungen vornehmen. Diese Spielerei hat einen sehr sinnvollen Hintergrund, da ja die computergesteuerten Maschinen in der Industrie nach solchen Prinzipien arbeiten. Der ATARI ST ist also theoretisch in der Lage, eine Werkzeugmaschine zu bedienen.

1.2.2 Die serielle Schnittstelle

Bei der seriellen Datenübertragung wird jedes Datenwort Bit für Bit übertragen. So benötigt man nur eine Datenleitung, jedoch ist diese Methode etwas komplizierter als die parallele Übertragung.

Die serielle Schnittstelle des ATARI ST trägt die Bezeichnung "Modem" und ist mit einem 25-poligen Stecker an der Rückseite herausgeführt. Es handelt sich um eine genormte RS-232- bzw. V.24-Schnittstelle, die im Gegensatz zum Centronics-Port bidi-

rektional, also in beiden Richtungen, arbeitet. An Pin 2 des Steckers, dem Datenausgang (TD = transmitted data), werden die HIGH und LOW-Signale als + bzw. -12 Volt ausgegeben.

Dieser Stecker ist zum Anschluß eines Telefonmodems vorgesehen, wie die Bezeichnung "Modem" bereits andeutet. Ein solches Modem stellt die Verbindung zwischen Rechner und Telefonleitung her, indem es die HIGH- bzw. LOW-Signale in zwei verschiedene Töne umsetzt. Falls Sie also jemandem, der ebenfalls einen Computer mit Telefonmodem besitzt, Daten (meist Texte) übermitteln wollen, so rufen Sie ihn einfach an. Wenn Sie dann beide Ihren Telefonhörer auf das Modem legen, in dem sich ein Mikrophon und ein Lautsprecher befinden, kann die Datenübertragung beginnen.

Um dem Empfänger jeweils zu signalisieren, daß ein Datenwort beginnt, wird am Anfang jedes Wortes ein sogenanntes Startbit übertragen. Die Leitung geht somit erst auf HIGH; dann folgen die Daten, abgeschlossen mit einem LOW-Signal als Stopbit. Die Geschwindigkeit, mit der sich das abspielt, wird Baud-Rate genannt. Diese Zahl (für akustische Modems 300) gibt die übertragenen Datenbits pro Sekunde an. Die Datenübertragung wird vom Betriebssystem des ATARI ST voll unterstützt, so daß Sie nur eine einmalige Einstellung der Schnittstelle vorzunehmen brauchen.

Wenn Sie niemanden kennen, der auch ein Modem besitzt, so können Sie auch eine sogenannte Mailbox anrufen. Diese zentralen Stellen der DFÜ (Daten-Fern-Übertragung) werden von vielen privaten und auch öffentlichen Anbietern bereitgestellt. Die Telefonnummern erfahren Sie am besten aus den einschlägigen Zeitschriften.

1.2.3 Diskettenanschluß

Der etwas zu groß geratene Stecker an der Rückseite des ATARI ST, mit der Bezeichnung *Floppy*, stellt den Anschluß für ein bis zwei Diskettenlaufwerke dar. Dieser wird vom Disk-Controller WD 1770 gesteuert, der die Befehle des Betriebssystems in Signale umsetzt. Wird z.B. das Inhaltsverzeichnis einer Diskette gewünscht, so fordert das GEMDOS den Controller auf, ihm die Daten des entsprechenden Sektors der gewählten Diskette zu übergeben. Der Controller hat dann viel zu tun:

1. Selektieren des gewünschten Laufwerks (1/2)
2. Einschalten des Motors, der die Diskette dreht
3. Einstellen des Schreib-Lesekopfes auf die Spur, die den Sektor enthält
4. Warten auf das Index-Pulse-Signal, welches einmal pro Umdrehung erfolgt und den momentanen Drehwinkel der Diskette erkennen läßt
5. nach Erreichen des Sektors Lesen und Umwandeln des analogen Signals vom Schreib-Lesekopf in digitale Signale
6. Weiterleiten der Signale

Dazu kommt noch, daß entschieden werden muß, auf welcher Seite der Diskette die Daten liegen, was allerdings nur bei doppelseitigen Laufwerken nötig ist. Dieses Signal kommt aber vom Sound-Chip (!). Das normale Floppy-Laufwerk SF 354 mit 500 KByte im unformatierten Zustand beschreibt die Disketten nämlich nur einseitig. Doppelseitig arbeitende Laufwerke werden jedoch voll unterstützt und sind direkt anschließbar. Die Belegung des Steckers ist wie folgt:

Pin	Benennung	Bedeutung
1 (30)	Read Data	Hier wird das gelesene Signal vom Schreib-Lesekopf herausgeführt
2 (32)	Side 0 Select	Wahl der Diskettenseite
3 (3)	GND	Masse (0V)
4 (8)	Index Pulse	pro Umdrehung der Diskette erfolgt hier ein Signal
5 (10)	Drive 0 Select	Anforderung von Disk 1
6 (12)	Drive 1 Select	Anforderung von Disk 2
7 (33)	GND	Masse
8 (16)	Motor on	Einschalten des Motors
9 (18)	Direction in	Festlegung der Schrittrichtung des Schreib-Lesekopfes
10 (20)	Step	einen Schritt nach innen/außen machen
11 (22)	Write Data	vom Rechner kommende Datensignale beim Schreiben
12 (24)	Write Gate	Erlaubnissignal zum Schreiben der Daten
13 (26)	Track 0	Meldung des Laufwerks, daß die äußerste Spur erreicht ist
14 (28)	Write Protect	Meldung, daß die Diskette schreibgeschützt ist

Diese Signale sind erfreulicherweise genormt und entsprechen denen von Laufwerken mit sogenanntem Shugart-Anschluß. Es ist somit recht einfach, andere Diskettenlaufwerke anzuschließen. Doch Vorsicht, eine falsche Verdrahtung könnte den Controller zerstören! Die äquivalenten Pin-Nummern der Shugart-Stecker sind in obiger Tabelle in Klammern beigefügt.

Vor der ersten Benutzung einer Diskette muß diese formatiert werden. Nach Wahl des Desktop-Menüpunktes *Formatieren* und Anklicken von OK zeigt sich, daß auch zweiseitige Laufwerke vorgesehen sind. Mit einem einseitigen Laufwerk dürfen Sie nie *Doppelseitig* wählen, wohl aber umgekehrt. Beim Formatieren werden die Spuren auf der Diskette gelöscht und einzelne Sektoren festgelegt. Alle Daten, die vielleicht vorher auf der Scheibe waren, sind damit gelöscht!

1.2.4 Die MIDI-Schnittstelle

An der Rückseite des ATARI ST befinden sich noch zwei kleine Stecker, die schon dadurch auffallen, daß es sich dabei um ganz normale 5-polige Diodenstecker handelt. Sie tragen die Bezeichnung MIDI-In und MIDI-Out und dienen zur Unterstützung von Musikinstrumenten wie z.B. Synthesizern. Wie geht das nun vor sich? MIDI bedeutet Musical Instrument Digital Interface und ist in vielen Synthesizern ebenfalls implementiert. Verbindet man nun den ST mit einer oder auch mehreren (bis zu 16) Orgeln, dann kann der ST zum Musiker werden, indem er entweder gespielte Melodien mitschneidet oder selbst die Instrumente bedient. Der Informationsaustausch erfolgt seriell nach dem gleichen Prinzip wie beim Modem-Port, nur daß die eingestellte Übertragungsgeschwindigkeit immerhin 31250 Bit pro Sekunde (Baud) beträgt.

Da mehrere Instrumente angeschlossen werden können, muß eine Unterscheidung der Geräte stattfinden. Dies geschieht nicht einfach durch eine Select-Leitung wie bei der Selektierung von Diskettenlaufwerken. Die Empfänger liegen alle parallel an denselben Drähten. Der Computer sendet aber zuerst ein Erkennungssignal, durch das sich nur jeweils ein Gerät angesprochen fühlt und allein das darauf folgende Kommando ausführt.

Nun ist diese schnelle Schnittstelle geradezu prädestiniert zur artfremden Verwendung. Weitere Rechner könnten beispielsweise über die MIDI-Leitungen mit dem ST verknüpft werden, um so ein Netzwerk aus bis zu 17 Computern zu bilden. Es müßte nur jeder dieser Nebenrechner eine eigene MIDI-Schnittstelle und eine Kennziffer besitzen, und der ST könnte jeden einzelnen ansprechen. Diese Kommunikation kann auch von BASIC aus vorgenommen werden, wenn nicht auf hohe Geschwindigkeiten Wert gelegt wird. Eine Ausgabe würde dann mittels des OUT 3,X-Befehls und der Empfang über den X=INP(3) abgewickelt werden. "X" ist hierbei der Wert des zu übertragenden Datenwortes.

1.2.5 Der ROM-Expansionsport

An der linken Seite des Rechners liegt etwas versteckt ein weiterer Stecker. Er trägt keine Bezeichnung, aber eine 40-polige Steckbuchse zur Aufnahme einer passenden Elektronikkarte. Eine solche Karte kann mit Festspeichern (ROM) bis maximal 128 KByte bestückt sein. Mögliche Anwendungen wären z.B. Betriebssystemerweiterungen, Hilfsprogramme, die einen Systemabsturz auffangen können, oder einfach Spiele. Die hohe Anzahl der Anschlüsse ist leicht zu verstehen, schließlich müssen der Speicherbereich adressiert und Daten übertragen werden können. Zur Adressierung von 128 KByte werden 17 Adressbits benötigt, und die Datenbreite beträgt nochmal 16 Bit.

An diesem Stecker fehlt scheinbar eine Schreib-Leseleitung, die auch die Ausgabe von Daten ermöglicht. Es ist aber nicht vorgesehen, sondern geradezu verboten, Daten in den Adressbereich des Steckers zu schreiben! Sie können es ja dennoch einmal probieren, falls Sie gerade nichts Wichtiges im Speicher haben. Der Anschluß liegt zwischen den Adressen $FA000 bis $FC000 (1024000-1032192). Wenn Sie z.B. POKE 1030000,0 tippen, wird das System neu starten und Sie werden sich im Desktop wiederfinden. Alle Daten im BASIC-Speicher sind damit gelöscht.

Der Erweiterungsstecker ist also nur für ROM-Karten erlaubt. Diese Einschübe nennt man Cartridges, ähnlich wie man sie z.B. von den alten ATARI 400/600/800-Rechnern her kennt (diese Karten funktionieren hier natürlich nicht!). Die ersten Daten auf den Karten enthalten Kennzahlen, die dem ATARI mitteilen, welche Art von Programm vorliegt. Ist das erste Langwort an der Adresse $FA0000 gleich $FA52255F, so handelt es sich um ein Diagnoseprogramm. Lautet es $ABCDEF42, dann ist ein Anwenderprogramm eingesteckt. Alle anderen Kombinationen werden ignoriert. Falls ein Diagnoseprogramm vorliegt, beginnt der Prozessor beim System-Reset fast sofort mit dessen Abarbeitung ab Adresse $FA0004. Anwenderprogramme müssen an dieser Adresse verschiedene Informationen über Anfang, Länge und Name des Programmes beinhalten.

1.2.6 Der Maus-/Joystickanschluß

Die zwei Stecker an der rechten Seite des ST sind zur Aufnahme von Joysticks vorgesehen. An Anschluß 0 kann gleichzeitig das Mauskabel eingesteckt werden. Der Unterschied zwischen Joystick und Maus liegt vor allem darin, daß der "Freudenknüppel" für alle vier Richtungen einen eigenen Signalpin hat, auf den er ein LOW-Signal (0V) legen kann. Die Maus dagegen gibt Richtungsimpulse an das System weiter, da sie ja mit verschiedenen Geschwindigkeiten bewegt werden kann.

Die Anschlüsse unterscheiden sich daher geringfügig. Zwar kann ein Joystick an beide Ports gesteckt werden, die Maus dagegen ist nur in Anschluß 0 funktionsfähig. Dieser Anschluß besitzt außerdem einen zusätzlichen Eingang, da die Maus über zwei Knöpfe verfügt.

Die Maus- bzw. Joystickstecker sind reine Eingänge. Eine Ausgabe von Signalen ist hier nicht möglich, da der zuständige Tastaturprozessor keinen Befehl zur Umkehr der Datenrichtung kennt. Es bestünde zwar die Möglichkeit, den Prozessor durch einen anderen zu ersetzen, dessen Betriebssystem solche zusätzlichen Befehle kennt, aber das ist nicht praktikabel. Ausgangssignale können daher wesentlich einfacher am Parallelport (Printer) ausgegeben werden, während sich für Input-Signale dieser Stecker ganz gut eignet. Die erlaubten Signalspannungen betragen 0V (LOW) und +5V (HIGH) und entsprechen dem sogenannten TTL-Pegel (TTL = Transistor-Transistor-Logik), der in der Digitaltechnik häufig verwandt wird. Will man beispielsweise einen Taster vom Computer abfragen, so braucht man ihn nur zwischen Masse (Pin 8) und einem der Pins 1-4 anzuschließen, so daß sich durch Abfrage des entsprechenden Joysticks eine Aussage über den Zustand der Taste machen läßt.

1.2.7 Der Monitorstecker

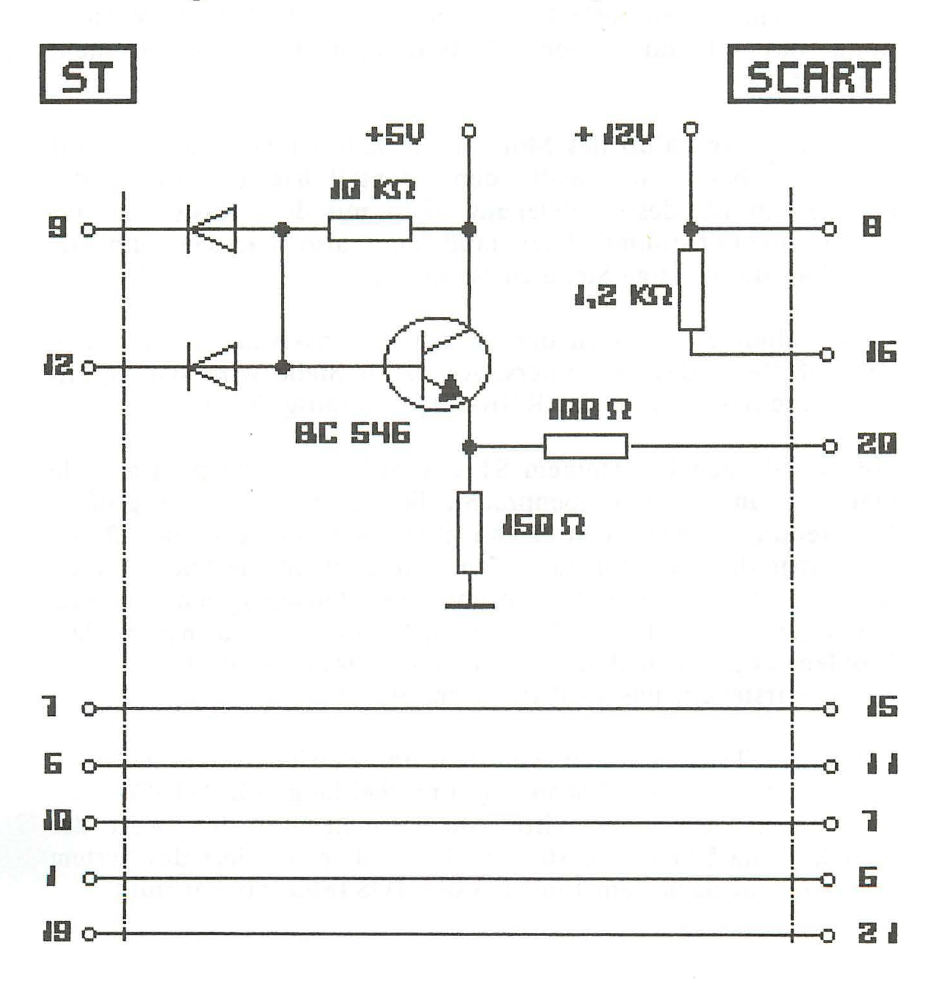
Was nützt einem der schönste Computer, wenn man nicht dessen momentane Tätigkeit sehen kann? Also muß eine Anschlußmöglichkeit für ein Datensichtgerät vorhanden sein, was beim ATARI ST selbstverständlich der Fall ist. Dieser Stecker mit der Aufschrift "Monitor" leistet allerdings mehr als das. Hier läßt sich nämlich nicht nur der Schwarz-Weiß-Monitor SM 124 anschließen, sondern der ST bekennt auch Farbe! In der Anschlußbelegung findet man die Begriffe Rot, Blau und Grün. Diese drei Signale bilden zusammen mit den Synchronisationsausgängen Horizontal- und Vertikal-Sync den sogenannten RGB-Anschluß und machen den ATARI ST farbfähig, wenn man einen Farbmonitor oder einen Fernseher mit SCART-Stecker besitzt.

Allerdings ist der Anschluß eines solchen TV-Gerätes nicht ganz unproblematisch, was nicht zuletzt an der merkwürdigen Buchsenform des Monitorsteckers liegt, der im Handel einfach nicht erhältlich ist. Außerdem sind die Signale, die der ST zur Verfügung stellt, nicht direkt für den Fernseher verwertbar. Zwar bietet der ATARI gleich zwei Synchronisations-Signale an, aber der SCART-Stecker verfügt nur über einen Sync-Eingang, so daß man mischen muß. Des weiteren benötigt der Fernseher zwei Schaltspannungen, um von Empfang auf RGB-Ebene umzuschalten, und zwar 12V und 2V. Diese Spannungen liegen jedoch nicht am Monitor-Stecker an.

Einen provisorischen Stecker kann man sich basteln, indem man in eine handelsübliche Lochrasterplatine Stecknadeln einlötet und diese als Lötnägel mißbraucht. Die Verdrahtung geschieht folgendermaßen:

ATARI ST		TV-SCART
1	Audio out	6 = Audio 2 links
6	Grün	11
7	Rot	15
10	Blau	7
8	Masse	17
9	HSync und	
12	VSync gemischt an	20 = Video-Eingang

Für die Mischung der Syncronisations-Signale reicht es aus, wenn Sie beide über je einen 10 KOhm-Widerstand an Pin 20 des SCART-Steckers bringen. Das ist allerdings nicht gerade elegant und belastet den ATARI mehr als nötig. Um eine korrekte Anpassung der Sync-Signale an den SCART-Stecker herzustellen, muß man schon ein wenig in die Elektronik-Kiste greifen. Man benötigt dazu einen Transistor, 4 Widerstände und zwei Dioden. Zur Stromversorgung dieser Schaltung werden 5 Volt benötigt, die man sich aber als Provisorium vom Pin 7 des Joystick-Steckers des ATARI ST holen kann. Die Schaltung kann ohne weiteres auf der als Stecker verwendeten Lochraster-Platine aufgebaut werden. Hier der Schaltplan:



Dieser Aufwand ist allerdings nur beim normalen ATARI 520ST nötig. Der große Bruder 520ST+ hat diese Zusammenschaltung der Sync-Signale nämlich bereits intus, wodurch sich allerdings die Steckerbelegung des Monitor-Anschlusses etwas ändert. Der Anschluß 8 führt hier nicht mehr Masse, sondern an ihm ist die 12 Volt-Spannung herausgeführt, die man zum Umschalten eines Fernsehers auf den Video-Betrieb braucht. Dieser Anschluß wird dann mit Pin 8 des SCART-Steckers verbunden. Der Fernseher geht dann beim Einschalten des Computers sofort in den Video-Mode über. Diese 12 Volt lassen sich zudem noch durch einen einfachen Spannungsteiler mit zwei Widerständen auf die noch benötigte Schaltspannung von 2 Volt herunterteilen, so daß man keinen zusätzlichen Anschluß mehr für die Verbindung des ST mit einem RGB-fähigen Fernsehgerät mehr benötigt.

Der Masse-Anschluß des Monitor-Steckers ist beim 520ST+ auf Pin 13 verbannt, was leider den Nachteil hat, daß dieser der einzige Pin ist, dessen Belegung nicht mit dem Raster unserer Platine übereinstimmt. Hier muß man also tricksen, um die Nadel an die richtige Stelle zu bekommen.

Und schließlich ist noch der Syncronisations-Anschluß umzulegen auf Pin 2 des Computers. An dieser Stelle wird das bereits gemischte Sync-Signal des RGB-Bildes herausgeführt.

Nun ergab sich bei meinem ST ein weiteres Problem. Das Bild erstrahlte in voller Farbenpracht, ließ sich aber trotz größter Überredungskünste nicht davon abbringen, in vertikaler Richtung über den Monitor zu laufen. Ein Blick in die Speicherzelle $448 = 1096 brachte die Lösung: das Betriebssystem glaubte, einen amerikanischen NTSC-Monitor vor sich zu haben. Das Problem ergab sich daraus, daß diese Geräte 60 Bilder pro Sekunde darstellen, unsere PAL-Norm aber nur 50.

Folgender Trick sei hier verraten. Im Betriebssystem befindet sich an der Adresse $5056 die Entscheidung, ob NTSC- oder PAL-Norm verwendet wird. Ändert man nun den Wert der Speicherzelle $5056 von $67 auf $66 und initialisiert das System neu, so steht nach dem Einladen des TOS.IMG das Farbbild.

Leider kann man diese Änderung nicht von der BASIC-Ebene aus, sondern nur mit dem SID-Programm des Entwicklungspaketes durchführen, da es nicht mit einem POKE abgetan ist. Schließlich muß man diese Änderung mit dem Betriebssystem abspeichern, um sie beim RESET mitladen zu können.

Wenn man das SID-Programm nicht besitzt, hilft hier ein anderer Trick. Man kann mit BSAVE ja einen Datensatz auf Diskette abspeichern. Ebenso kann diese Datei auch ein Programm sein, das direkt ausführbar ist. Und genau das wollen wir nun machen. Das folgende Programm erstellt aus Daten ein Programm und legt es auf der Diskette ab. Dieses Programm kann vom Desktop aus aufgerufen werden und ändert dann die Vertikal-Frequenz des Farbausganges auf 50 Hz, eben den Wert der PAL-Norm. Der Programmname SETPAL.PRG ist hier zwar anschaulich, aber es kann auch von Vorteil sein, es AASETPAL.PRG zu nennen. Bei diesem Namen steht das Programm an erster Stelle im Inhaltsverzeichnis, sofern nach Namen sortiert wurde. In Anbetracht der Tatsache, daß beim Aufruf dieser Routine das Bild ständig läuft, ist sie so wesentlich leichter mit dem Mauszeiger zu treffen!

```
10    rem *** Create SETPAL.PRG ***
20    for i=1 to 74  step 2        : rem Schleife für 36 Worte
30    read a                       : rem Wort lesen
40    poke i+1999,a                : rem im Speicher ablegen
50    next i                       : rem bis fertig
60    bsave "SETPAL.PRG",2000,74  : rem Abspeichern
70    end
90    rem  --- Programm - Daten ---
100   data 24602,0,40,0,0,0,0,0,0,0,0,0,0,0
110   data 17063,16188,32,20033,-8196,0,6
120   data 5116,2,-1,-32246
130   data 12032,16188,32,20033,-8196,0,6
140   data 16999,20033,0,4,0,0
```

Bevor Sie das Programm starten, vergewissern Sie sich bitte, ob Sie die Daten exakt abgetippt haben. Ein einziger kleiner Fehler führt mit größter Wahrscheinlichkeit zum Absturz!

Das erstellte Maschinenprogramm setzt sich folgendermaßen zusammen:

Die Daten in Zeile 100 bilden den Programmkopf, den das TOS für die Ausführung des Programmes benötigt. Er enthält einige Aussagen über die Größe und Form des Gesamtprogramms, was für uns hier nicht wichtig ist. Der Teil in Zeile 110 schaltet den Prozessor in den Überwachermodus, wo die vorzunehmende Änderung nur machbar ist. Danach erfolgt der Eingriff in den I/O-Bereich des Programmspeichers, wo die Vertikalfrequenz bestimmt wird. Schließlich wird das System wieder in den User-Modus geschaltet und zum Desktop zurückgeführt. Wie dies alles für den Prozessor aussieht, werden wir uns später im Abschnitt über die Maschinensprache des MC 68000 ansehen. Bleiben wir vorerst noch bei der Betrachtung des ST-Aufbaus.

Anhand dieses Beispiels können wir gleich eine weitere Möglichkeit betrachten, die der ATARI ST bietet. Um unser Beispielprogramm direkt beim Einschalten des Rechners arbeiten zu lassen, brauchen wir nur einen weiteren Ordner auf unserer Systemdiskette anzulegen. Dieser Ordner muß den Namen AUTO tragen und veranlasst den Computer, nach dem Einladen des Betriebssystems sofort alle in ihm enthaltenen Programme zu starten. Hier kann also z.B. auch der BASIC-Interpreter liegen, der ebenfalls von selbst startet.

1.3 Die intelligente Tastatur

Wie bereits erwähnt, hat der in der Tastatur eingebaute Prozessor viele Aufgaben. Er entlastet durch ständige Überwachung von Maus und Tastatur den MC 68000 des "Chefrechners" in großem Maße. Durch einen eigenen Speicher ist die Tastatur somit in der Lage, die Zustände von Maus, Tasten, Joysticks und auch der Uhr ständig parat zu haben. Alle Informationen erhält der Hauptrechner nur bei entsprechender Anfrage. Er

sendet dann seriell ein Kommando und evtl. einige Parameter an den Tastaturprozessor, worauf dieser die gewünschte Information zurückgibt. Dies geschieht mit einer Geschwindigkeit von 7812.5 Bit pro Sekunde. Der Tastaturprozessor ist so programmiert, daß er folgende Befehle versteht:

1.3.1 Befehlsübersicht

Kommando	Reaktion
7	Einstellung der Reaktion auf Drücken der Maustasten
8	Ab sofort Meldung der relativen Mausposition
9	Ab sofort Meldung der absoluten Mausposition
10	Mausbewegung als Cursortastendruck melden
11	Verzögerung der Meldung von Mausbewegungen einstellen
12	Skala für Mausposition einstellen
13	Abfrage der absoluten Mausposition
14	Setzen des internen Koordinatensystems
15	Y-Ordinatenursprung liegt unten
16	Y-Ursprung liegt oben
17	Datenübertragung wieder aufnehmen (nach #19)
18	Abschalten der Maus
19	Datenübertragung stoppen
20	Ab sofort Meldung jeder Joystickbewegung
21	Abschalten der Funktion 20 sowie der Maus
22	Joystick-Abfrage
23	Ab sofort Joystick-Dauermeldung
24	Ab sofort Feuerknopf-Dauermeldung
25	Ab sofort Meldung der Joystickbewegungen als Cursortastendruck (Joystick 0)
26	Joystick abschalten
27	Uhrzeit setzen
28	Uhrzeitabfrage
32	Tastaturspeicher laden
33	Tastaturspeicher lesen
34	Start eines Programmes im Tastaturspeicher

Außerdem versteht er diverse Abfragen nach dem momentanen Zustand der Einstellungen.

Doch sehen wir uns mal einige Beispiele für die Anwendung der Tastaturbefehle an. Der Haken bei der Programmierung liegt dabei in unterschiedlicher Zahl und Bedeutung der zu übergebenden Parameter, die für die einzelnen Funktionen von Bedeutung sind. Außerdem sind die Ergebnisse der Funktion nicht immer leicht zu erhalten. Betrachten wir dafür ein Beispiel. Die Funktion 22 veranlaßt den Tastaturprozessor, den momentanen Zustand des Joysticks 1 auszugeben. Das Ergebnis liegt danach in Speicherzelle 3591. Ein Programm für diese Abfrage sieht dann so aus:

1.3.2 Joystickabfrage

```
10  rem *** Joystick 1 - Abfrage ***
20  out 4,22                          : rem Funktionsaufruf
30  defseg = 1
40  js = peek(3591)                   : rem Ergebnis holen
```

Durch den OUT 4,n-Befehl des ATARI-BASIC wird die Tastatur veranlaßt, den momentanen Stand des Joysticks 1 abzufragen und an den Hauptrechner weiterzumelden. Das Betriebssystem legt diesen Wert dann in Speicherzelle 3591 ab, wo er durch PEEK(3591) augelesen werden kann. In der so erhaltenen Zahl ist die Joystickstellung binär enthalten. Die Bits 0 und 1 geben hier die vertikale Bewegung des Joysticks an, Bit 2 und 3 stehen für die horizontale Richtung. Wie man sieht, braucht dieser Funktion kein Parameter übergeben zu werden. Dies wird durch das nächste Beispielprogramm vorgeführt:

1.3.3 Maus als Cursorsteuerung

```
10   rem *** Mausbewegung als Cursortasten ***
20   out 4,10          : rem Befehl senden
30   out 4,10          : rem X-Verzögerung
40   out 4,15          : rem Y-Verzögerung
```

Hier wird die Tastatur angewiesen, Bewegungen der Maus als Betätigung der Cursortasten auszugeben. Diese Betriebsart ist bei der Editierung von BASIC-Programmen sehr nützlich, da man mit der Maus den Cursor recht schnell über den Bildschirm bewegen kann. Die Verzögerungswerte, die in den Zeilen 30 und 40 übergeben werden, bedeuten die horizontale bzw. vertikale Bewegung der Maus für einen Cursorschritt. Hier ist es also nicht zu empfehlen, große Werte einzugeben, wenn man nur einen mäßig großen Schreibtisch hat. Um die Maus wieder als Zeiger zu sehen, muß der Befehl OUT 4,8 gegeben werden. Wie man sieht, sind die Übergabewerte einfach als weitere OUT-Anweisungen eingegeben worden. Dabei muß man sehr genau darauf achten, daß die richtige Parameteranzahl eingehalten wird. Andernfalls könnte ein vermeintlicher Parameter z.B. die Datenübertragung zwischen Tastatur und Computer stoppen.

1.3.4 Uhrzeit- und Datumsverwaltung

Kommen wir nun zu einem Beispiel, in dem Parameter sowohl ein- wie auch ausgegeben werden. Es handelt sich dabei um die Verwaltung der in der Tastatur eingebauten Uhr, die für manche Anwendungen sicher interessant ist.

```
10   rem *** Uhrzeit auslesen ***
20   out 4,28                 : rem Uhrzeitanfrage
30   for i = 3584 to 3589     : rem Uhrzeit / Datum-Puffer
40   x = peek(i) and 255      : rem Doppelziffer lesen
50   a$=a$ + str$(int(x/16)) : rem erste Ziffer
60   a$=a$ + right$(str$(x and 15),1)
```

```
70   a$=a$ + "."
80   next i
90   print "Datum   : " ; left$(a$,12)
100  print "Uhrzeit : " ; right$(a$,12)
110  end

130  rem *** Uhrzeit / Datum setzen ***
140  out 4,27            : rem Set Time- Befehl
150  print "Bitte Datum / Uhrzeit eingeben ( JJMMTTSSMMSS )"
160  input a$            : rem Eingabe
170  for i = 1 to 6      : rem BCD - Berechnung
180  d =0
190  d =d + asc(mid$(a$,i*2,1))-48
200  d =d + (asc(mid$(a$,i*2-1,1))-48)*16
210  out 4,d             : rem Parameter setzen
220  next i
230  a$="" :goto 10      : rem neue Zeit ausgeben
```

Eigentlich handelt es sich hierbei um zwei getrennte Programme. Der Aufruf von Teil 1 bewirkt die Ausgabe des aktuellen Datums inclusive Uhrzeit. Die Reihenfolge ist dabei Jahr, Monat, Tag, Stunde, Minute, Sekunde. Die Verwendung des sich ergebenden Formates in der Textvariablen a$ ist dabei völlig Ihnen überlassen. Außerdem kann durch Änderung der END-Anweisung in Zeile 110 in RETURN die Funktion als Unterroutine aufgerufen werden. Am kompliziertesten gestaltet sich bei der Auswertung der übernommen Daten die Umwandlung der verwendeten BCD-Form (BCD = Binary Coded Decimal) in einzelne Ziffern. In dieser Kodierung werden nämlich für jede Datumsziffer nur 4 Bit eines Datenbytes belegt. Somit wäre z.B. der Wert der BCD-Zahl 31 eben Hexadezimal $31, was wiederum einer dezimalen 49 entspricht (siehe auch Kapitel 2.1).

Dasselbe Problem taucht nun auch bei der entgegengesetzten Umwandlung von Einzelziffern in BCD-Zahlen auf. In Zeile 130 beginnt die Routine zum Setzen des Datums. Nach Aufruf des "Set Time"-Befehls 27 erfolgt dann die Eingabe der Zeit in der Reihenfolge wie angegeben, z.B. 053010192015 für den 30.10.1985 um 19 Uhr 20 und 15 Sekunden. Dabei wird die 8 im

Jahr automatisch addiert. Nach dieser Eingabe wird nun jeder Eintrag, umgewandelt in BCD-Zahlen, der Tastatur übergeben. Der darauffolgende Sprung zur Datumsausgabe dient nur der Kontrolle der richtigen Funktionsausführung und kann entfallen.

1.3.5 Tastenabfrage

Eine nicht ganz unwichtige Funktion der Tastatur wollen wir auch noch betrachten, nämlich die Tasten selbst. Das Tastenfeld wird von dem eingebauten Prozessor ständig überwacht. Wird eine Taste gedrückt, so legt er den Wert dieser Taste in seinem eigenen Speicher ab. Dieser Speicher hat den mageren Umfang von 128 Byte und enthält zusätzlich noch viele andere aktuelle Informationen, aber er reicht aus. Der Hauptrechner, bzw. das Betriebssystem, läßt sich zyklisch die aktuellen Daten mitteilen. Die gedrückten Tasten werden dann im Hauptspeicher abgelegt und bei Bedarf ausgelesen.

Will man von einem BASIC-Programm aus eine Taste abfragen, so gibt es hierfür mehrere Möglichkeiten. Man kann mit der Anweisung X=INPUT$(1) auf einen Tastendruck warten und das Ergebnis in X dann weiterverwenden. Diese Funktion gibt aber nur den ASCII-Code der normalen alphanumerischen Tasten zurück. Viel umfangreicher ist dagegen die Information, die man durch den Befehl X=INP(2) erhält. Der X-Wert, den man auf diese Weise bekommt, enthält den Tastencode von genau der Taste, die gedrückt wurde. Damit sind auch die Funktionstasten gemeint, die die Werte 187-196 haben. Will man auf die Betätigung einer Funktionstaste warten, so kann man das mit diesem Programmteil bewerkstelligen:

```
100  print "Bitte Funktionstaste drücken !"
110  x = inp(2)                           : rem Taste holen
120  if x < 187 or x > 196 then 110       : rem falsche Taste
130  ft = x - 186                         : rem F-Nummer
140  on ft goto f1,f2,f3,f4,f5,f6,f7,f8,f9,f10
```

Dieses Programm verzweigt nach Betätigung einer Funktionstaste zu dem gewählten Programmteil, das natürlich auch einen anderen Namen haben kann als F1 bis F10. Diese Art der Programmierung wird üblicherweise zur Menüsteuerung von Programmabläufen verwendet, doch dazu später.

1.4 Die Maus

Die handliche graue Maus mit den zwei Tasten stellt ein hervorragendes Eingabemedium dar. Das ständige Hin- und Herblicken zwischen Tastatur und Bildschirm wird überflüssig, da man, zumindestens bei Menübedienung, quasi auf den Bildschirm tippt. Jede Bewegung der Hand wird dabei in X- und Y-Schritte mit einer Auflösung von vier Schritten pro Millimeter registriert. Wie funktioniert das?

Nun, ein Blick in das Innere des Maus-Gehäuses läßt sch on einige Vermutungen zu. Nimmt man durch Verschieben der Blende auf der Mausunterseite die Kugel heraus, so sieht man drei kleine Metallrollen. Die schräge Rolle dient nur der Stabilisierung der Kugel, die anderen beiden geben die entsprechende Rollbewegung über je eine Lochscheibe weiter, an der wiederum zwei Lichtschranken sitzen.

Der in der Tastatur befindliche Prozessor HDV 6301 V1 wertet nun alle vier Impulsfolgen aus. Angenommen, die Maus wird nach vorne, also in Y-Richtung, bewegt, dann erhält der Prozessor an den Y-Puls-Eingängen zwei gegeneinander verschobene Impulsfolgen. Er wartet dabei erst einmal ab, bis beide Richtungssignale logisch 1, also HIGH sind, und erkennt die Schieberichtung daran, welches der beiden Signalen zuerst auf LOW-Pegel fällt. Derselbe Vorgang spielt sich gleichzeitig für die X-Verschiebung ab, so daß die wirkliche Richtung der Maus erkannt wird.

Der Prozessor in der Tastatur gibt sowohl die Richtung der Mausbewegung, als auch die betätigte Taste, Joystickstellungen, Uhrzeit und Datum auf Anfrage des Betriebssystems seriell zurück. Die Übertragung erfolgt mit etwa 7800 Bit pro Sekunde

(Baud). Um diese Datenübermittlung braucht sich der Programmierer jedoch glücklicherweise nicht zu kümmern, da er von einigen Programmiersprachen bei der Anfrage der Mausposition vollständig unterstützt wird. Das mit dem Rechner zusammen gelieferte DR LOGO bietet dazu den MOUSE-Befehl, der eine für LOGO typische Liste zurückgibt.

1.4.1 Die Maus als Malstift

Ein Beispielprogramm zum Malen auf dem Bildschirm (Grafikfenster) sieht dann so aus:

```
TO DRAW
  IF (ITEM 4 MOUSE) [SETPOS MOUSE]
  DRAW
  END
```

Dieses Programm wird im LOGO-Editor eingegeben und nachher von der Kommandoebene aus mit DRAW aufgerufen. Es handelt sich um ein rekursives Programm, d.h. es ruft sich selbst wieder auf. Dies geschieht durch den Befehl DRAW am Ende der Prozedur. Die zweite Zeile zeigt die Verwendung der Listenstruktur des LOGO-Befehls MOUSE. Dies ist am einfachsten als Variable anzusehen, deren Werte hintereinander in einer Liste stehen. Die Reihenfolge der einzelnen Einträge in dieser Liste ist [X Y B1 B2 B3], wobei X und Y die momentanen Koordinaten des Mauszeigers angeben. B1 und B2 bedeuten den Status der beiden Tasten auf der Maus, B1 für links und B2 für rechts. Sie sind wahr (TRUE), wenn die entsprechende Taste gedrückt ist. B3 gibt schließlich an, ob sich der Mauszeiger im Graphikfenster des LOGO befindet (TRUE wenn ja). In obigem Beispielprogramm wird durch (ITEM 4 MOUSE) der vierte Eintrag der Liste, also die rechte Taste, verwendet. Dadurch wird eine Linie zum Mauszeiger hin gezeichnet, wenn die rechte Taste gedrückt wird.

Beim Ausprobieren des Programmes werden Sie sicher bemerken, daß nichts passiert, wenn Sie während der Betätigung der Taste die Maus bewegen. Dieser Effekt tritt auch in BASIC-

Programmen auf. Der Grund hierfür liegt im GEM, welches bei Mausbewegungen ständig den Zeiger neu zeichnen muß und daher den Rechner solange beansprucht. Dieser meist unerwünschter Nebeneffekt kann aber auch brauchbar sein, da man ein laufendes Programm durch Herumrollen der Maus vorübergehend stoppen kann, um beispielsweise aktuelle Ausgabewerte zu lesen.

Nun ist ein solches Programm aber nicht in jeder Sprache so verblüffend einfach wie in DR LOGO. Die beliebte Programmiersprache BASIC bietet für die Mausabfrage keinen Befehl Hier muß man sich also manchen Klimmzug abringen, und zwar muß dies mittels des VDI im GEM abgewickelt werden (siehe Abschnitt 3.2: GEM).

Eine Bemerkung für Besitzer eines alten ATARIs oder eines ähnlichen Heimcomputers: der Stecker am Mäuseschwanz passt auch in jeden Joystickstecker. Durch ein geeignetes, sehr schnelles Maschinenprogrammes läßt sich somit die Maus auch an Ihrem alten Rechner anschließen und sinnvoll einsetzen. Die Abfrage der Eingänge muß dabei etwa 1000 mal pro Sekunde geschehen, da diese ihren Zustand ja schon bei einer Verschiebung von $\frac{1}{4}$ Millimeter ändert!

2. Datengedächtnisse

Alle eingegebenen Daten bzw. Programme müssen natürlich irgendwie gespeichert werden. Hierzu gibt es in der Computertechnik verschiedene Methoden, wie z.B. elektronische Halbleiterspeicher und Magnetplatten. Die Fähigkeiten dieser Datenschränke wollen wir nun betrachten.

2.1 Der interne Speicher

Der interne Speicher eines Computers besteht aus zwei verschiedenen Typen:

- dem Schreib/Lesespeicher (RAM = random access memory)
- dem Festwertspeicher (ROM = read only memory)

In den ATARI 520ST und den 260ST ist ein RAM von 512 KByte eingebaut, was genau 524288 Bytes sind. Der ATARI 520ST+ enthält sogar das doppelte, nämlich 1024 KByte. Es handelt sich um elektronische Bauelemente, die ihren Inhalt nur bei ausreichender Stromversorgung behalten. Fällt also einmal bei der Arbeit der Strom aus, so ist der gesamte interne Speicherinhalt verloren. Um sich gegen diesen zwar seltenen, aber um so ärgerlicheren Unfall abzusichern, ist es empfehlenswert, während des Programmierens hin und wieder das bereits geschriebene auf Diskette abzuspeichern.

In diesem für Heimcomputerkenner riesigen Speicher liegen nun alle Daten und Programme, die der Computer zum Funktionieren braucht. In der vorliegenden Version des ATARI ST wird auch das Betriebssystem des Rechners noch ins RAM geladen und belegt davon ca. 200 KByte. Dazu kommt dann der Bildschirminhalt, der wiederum 32 KByte schluckt, sowie einige Werte und Tabellen für das Betriebssystem. Wenn jetzt auch noch z.B. die Programmiersprache LOGO geladen wird, die ca. 110 KByte lang ist, ist vom RAM nicht mehr allzuviel übrig.

Wie der Name schon sagt, läßt sich dieser Schreib/Lesespeicher beliebig verändern. Die oben erwähnten Daten, die das Betriebssystem zum Speichern von Systemparametern verwendet, sind also manipulierbar, was eine große Herausforderung für alle Programmierer bedeutet, die über eine normale Programmierung hinaus das System voll ausschöpfen wollen. Die BASIC-Befehle PEEK und POKE sind hierfür vorgesehen, doch vor dem Ausprobieren sollten Sie erst genau wissen, was in welchen Speicherzellen steht.

Merke: Manipulation einer unbekannten Speicheradresse kann völlig unerwartete Effekte haben, die sich vielleicht erst zu einem späteren Zeitpunkt negativ bemerkbar machen!

Bevor wir also zum Experiment schreiten, wollen wir erst mal den Inhalt des ATARI ST eingehender betrachten.

Von den variablen Speicherzellen kommen wir nun zu den unveränderlichen, den ROMs. Dies sind fest programmierte Bauteile, deren Inhalt auch ohne angelegte Betriebsspannung gleich bleibt. In der neuen Version des ATARI ST wird das Betriebssystem TOS in einem solches ROM eingebaut sein, so daß es nach dem Einschalten sofort verfügbar ist. Im vorliegenden Rechner der ersten Serie enthält das ROM lediglich ein Programm, welches das TOS von der Diskette lädt. Einziger Vorteil dieser Technik ist die Manipulierbarkeit des Betriebssystems selbst.

Schauen wir uns nun den gesamten Speicherbereich des ATARI ST an. Da es sich um einen Rechner mit einem 24 Bit breiten Adreßbus handelt, kann er direkt 2^{24}, also 16 Megabyte adressieren. Dieser enorme Adreßbereich ist natürlich nicht vollständig mit Speicherbausteinen belegt. Eine folgende Tabelle zeigt die Speicherbelegung des Rechners:

2.1.1 Adressenbelegung des ATARI ST

Adresse	Belegung	Beschreibung
00 0000	ROM	Reset: Supervisor Stack Pointer
00 0004	ROM	Reset: Init Vektor
00 0008	RAM	RAM Anfang
07 FFFF	RAM	RAM-Top bei 512 KByte
0F FFFF	RAM	RAM-Top bei 1 MByte (520ST+/1040STF)
1F FFFF	RAM	RAM-Top bei 2 MByte
3F FFFF	RAM	RAM-Top bei 4 MByte (Maximum)
40 0000 F9 FFFF	unused	unbelegter Bereich
FA 0000 FB FFFF	ROM	ROM- Einschub (128 KByte)
FC 0000 FE FFFF	ROM	ROM: Betriebssystem (192 KByte)
FF 0000 FF 7FFF	unused	unbelegter Bereich
FF 8000 FF 8800	I/O	I/O-Bereich (2 KByte)
FF A000 FF BFFF	I/O	I/O-Bereich (2 KByte)
FF C000 FF FFFF	unused	unbelegter Bereich

Wie Sie sehen, ist der Bereich von 00 0000 bis 07 FFFF mit RAM belegt, wobei eine Erweiterung auf bis zu 4 Megabyte vom Betriebssystem zugelassen ist und für Bastler wohl kein Problem darstellt. Das Betriebssystem selbst liegt dann (später) im ROM, das an der Speicherzelle FC 0000 beginnt. Weiter "oben" sind die Bereiche der I/O-Bausteine (siehe auch Abschnitt 1.1). In diesen Speicherabschnitten liegen die Adressen, in denen die Parameterübergabe mit den Sonderbausteinen abgewickelt wird, und die vom Rechner als privilegiert angesehen werden; ein Zugriff vom Benutzer wird sofort durch Programmabbruch bestraft, einem sogenannten BUS-ERROR, der dann zur Ausgabe der berühmt-berüchtigten Rauchpilze auf dem Bildschirm (2 Stück) und zur Rückkehr in das GEM-Desktop führt. Dieser I/O-Bereich ist folgendermaßen belegt:

2.1.2 Die Adressen der I/O-Chips

ab FF 8000	- Speicherkonfiguration
ab FF 8200	- Register für den Video-Chip
ab FF 8600	- DMA/Disk-Controller
ab FF 8800	- Register des Sound-Chips
ab FF FA00	- Timer und Interrupt-Chip
ab FF FC00	- Tastatur- und MIDI-Chips

Noch ein wichtiger Bereich ist zu erwähnen: das erste KByte ab Adresse 000000. Hier liegen so wichtige Systemparameter, daß der normale Benutzer auch diesen Adreßbereich nicht manipulieren darf (sonst sieht man wieder die oben erwähnten Rauchpilze!). Es handelt sich bei diesen Parametern um sogenannte Vektoren, d.h. in diesen Speicherplätzen befinden sich Sprungadressen, wohin der Prozessor beim Auftreten von Programmunterbrechungen verzweigt. Diese Unterbrechungen werden Exceptions (Ausnahmen) genannt und treten aus unterschiedlichen Gründen auf. So ist z.B. auch der Zugriff auf geschützte Speicherbereiche (I/O oder auch diese Vektorentabelle) ein Anlaß für den Computer, einen dieser Exception-Vektoren zu verwenden. Die Tabelle ist folgendermaßen organisiert:

2.1.3 Fehler-Vektoren

Nr.	Adresse	Verwendung bei
	$000	Reset: initial SSP
	$004	Reset: initial PC
2	$008	Bus Error (s.o.)
3	$00C	Address Error
4	$010	Illegaler Befehl
5	$014	Division durch Null
6	$018	CHK - Befehl
7	$01C	TRAPV - Befehl
8	$020	Privilegsverletzung
9	$024	Trace
10	$028	Axxx - Befehlsemulation
11	$02C	Fxxx - Befehlsemulation
	$030-$038	Reserviert
	$03C	Uninitialisierter Interrupt
	$040-$05F	Reserviert
	$060	Unberechtigter Interrupt
	$064-$083	Level 1-7 Interrupt
	$080-$0BF	TRAP-Befehle
	$0C0-$0FF	Reserviert
	$100-$3FF	Interrupt-Tabelle:
	$100	Parallel-Port intern
	$104	RS232 Carrier Detect
	$108	RS232 Clear to send
	$10C	unbenutzt
	$110	unbenutzt
	$114	200 Hz System Clock
	$118	Tastatur/MIDI-Interrupt
	$11C	unbenutzt
	$120	HSync
	$124	RS232 Sende-Fehler
	$128	RS232 Sendepuffer leer
	$12C	RS232 Empfangs-Fehler
	$130	RS232 Empfangspuffer voll

Nicht jeder Vektor wird bei einem Fehler verwendet, aber dennoch sind viele der Anlässe für einen Interrupt unangenehm für unser laufendes Programm. Die laufende Nummer der Vektoren entspricht bei einem Systemabsturz genau der Anzahl an Bömbchen auf dem Bildschirm, so daß man durch bloßes Abzählen auf den aufgetretenen Fehler schließen kann. Wenn ein solcher Absturz stattgefunden hat, versucht das Betriebssystem, zu retten, was zu retten ist. Oft klappt das zwar nicht, und es erfolgt wieder einmal der Griff zum Reset-Taster. Sollte es aber möglich sein, den Betrieb wieder aufzunehmen, so braucht der Rechner die Informationen über den Betriebszustand vor dem Auftreten des Fehlers. Diese sind auch vom Betriebssystem vor dem Absturz in den folgenden Speicherplätzen abgelegt worden:

$0380	=$12345678, wenn die Daten gültig sind
$0384	ab hier liegen die geretteten D0-D7-Register
$03A4	und ab hier die A0-A6-Adreßregister
$03C0	der alte Supervisor-Stapelzeiger A7
$03C4	enthält die Nummer des aufgetretenen Fehlers
$03C8	hier liegt der alte User-Stapelzeiger A7
$03CC	und 16 Worte vom alten Supervisor-Stapel

Die Art der Erklärungen der Vektoren und Datentypen wird Ihnen vielleicht unbekannt vorkommen, da es sich hier um Vorgänge auf Maschinensprache-Ebene handelt. Für die Erklärung dieser Vorgänge sowie deren Nutzung zu eigenen Zwecken in der Maschinensprache-Programmierung verweise ich auf das ATARI ST-Intern Buch, das ebenfalls bei DATA BECKER erhältlich ist. Aber die Frage stellt sich zuerst: Was sind denn diese Vektoren überhaupt? Wollen wir uns das einmal näher ansehen.

2.1.4 Zeiger

Ein Vektor, auch Zeiger genannt, ist ein sehr wichtiger Bestandteil von Computersoftware. Es handelt sich um Speicherzellen, in denen wiederum eine Adresse steht. Ein Beispiel hierfür ist die Systemvariable $44E. In diesem Speicherlangwort steht die Adresse des Bildschirmspeichers, der im oberen Speicherbereich liegt. Das BIOS oder auch das GEM erfährt nur

durch Zugriff auf diesen Zeiger die Lage des Video-RAMs, das ja in einem ATARI ST mit 520 KByte an einer anderen Stelle liegt als in einem ATARI 520ST+ mit 1 MByte. Darin liegt der große Vorteil der Zeigertechnik. Man kann ein und dasselbe Programm durch den Umweg über einen Vektor auf Speicherbereiche zugreifen lassen, deren Lage variabel ist.

Nehmen wir unser Beispiel des Bildschirmzeigers. Um auf den Bildschirm zuzugreifen, brauchen wir nur den Vektor auszulesen und als Adressierung zu verwenden.

```
10   rem *** Bildpunkte setzen mit POKE ***
20   a =peek(1102) *65536 +peek(1104)
25   if peek(1104)<0 then a =a +65536
30   z =20                           : rem Bildzeile
40   s =5                            : rem Bildspalte
50   bz =peek(10556)                 : rem Byte pro Zeile
60   poke a +s +z *bz,255            : rem Strich setzen
```

Die Zeilen 20 und 25 machen die Schwierigkeit der Zeigerverarbeitung klar. Der Adressbereich des ATARI ST geht ja weit über 64 KByte hinaus, so daß ein Zeiger mehr als 16 Bit haben muß. Es handelt sich dabei also um 32 Bit-Adressen, sogenannte Langwörter, deren Zusammensetzung dem BASIC-Programmierer überlassen bleibt. Das erste Wort des Vektors steht bei obigem Beispiel in dem Wort ab 1102 und hat für das gesamte Langwort die Wertigkeit von 65536, d.h. 2^{16}. Daher muß dieses höherwertige Wort mit 65536 multipliziert werden. Dazu addiert man dann den Wert des niederwertigen Wortes und erhält so die ganze Zeigeradresse. Dieser Umweg ist glücklicherweise unnötig, da das Personal-BASIC auch die Verarbeitung von Langworten beherrscht. Einfacher lautet es somit:

```
20   defdbl a : a=1102      : rem Adresse des Zeigers
25   a = peek(a)            : rem direkt ein Langwort!
```

Das Betriebssystem benötigt ebenfalls keinen Umweg, da der Prozessor direkt mit Langwörtern arbeiten kann. Die Zusammensetzung der gewünschten Bildschirmadresse läuft aber auch im Rechner wie in den Zeilen 50 und 60 ab, wenn etwas auf das

Bild geschrieben werden soll. Schließlich ist ja auch ein Buchstabe eine Zeichnung, die aus 16 * 8 Punkten zusammengesetzt wird. Ein Zeichen auf den Bildschirm zu schreiben, läßt sich auch auf diese Weise bewerkstelligen. Ändern wir das obige Programm doch etwas ab:

```
60   for i = 0 to 10            : rem Schleifenbeginn
70   read x$                    : rem Musterzeile lesen
80   x=0 : for j = 1 to 10      : rem Auswertung
90   x= x-(mid$(x$,j,1)="*") * 2^10-j
100  next j
110  poke a+s+(z+i)*bz,x        : rem Zeile setzen
120  next i                     : rem und weitermachen
130  end
135  rem --- Musterdaten ---
140  data "          "
150  data "    **    "
160  data "   ****   "
170  data "  ******  "
180  data " ** ** ** "
190  data "**********"
200  data "** **** **"
210  data "**  **  **"
220  data " **    ** "
230  data "  ******  "
240  data "    **    "
```

Nun haben wir das Betriebssystem umgangen und einen Smiley auf den Bildschirm gezaubert. Es hätte auch diesen Ablauf der graphischen Darstellung verfolgt und auf den Zeiger zugegriffen. Doch was passiert, wenn man den Wert des Vektors ändert? Probieren wir doch dieses "Verbiegen" des Zeigers einfach aus und geben ein:

poke 1104,peek(1104)+1600 (Return)

Der Bildschirm "springt" nach unten, was wir durch Anklicken irgendeines Fensters feststellen können. Das Betriebssystem wird mit der falschen Zeigeradresse irregeführt und hält den falschen Speicherbereich für den Bildspeicher! Doch machen wir das

lieber wieder rückgängig, da sich sonst unangenehme Effekte ergeben.

poke 1104,peek(1104)-1600 (Return)

Und die Welt ist wieder in Ordnung. Doch gibt es auch viele Zeiger, deren Verbiegung nicht so glimpflich abgeht wie der Bildschirm-Vektor. Diese Zeiger repräsentieren Sprungadressen (sie zeigen auf Maschinensprache-Routinen) zu denen der Prozessor verzweigen soll. Eine sehr wichtige Anwendung dieser Art Wegweiser sind die Interrupt-Vektoren. Interrupts sind Programmunterbrechungen, die aus verschiedenen Gründen immer wieder auftreten. Stellt man einen solchen Zeiger auf eine Adresse, in der sich kein gültiges Maschinenprogramm befindet, so steigt der Rechner mit den üblichen Abschiedsgrüßen in Form von Bömbchen aus. Da diese Interrupts meist alle Bruchteile von Sekunden erfolgen, tritt dieser Absturz fast augenblicklich auf. Also lassen wir diese Vektoren besser in Ruhe!

Nehmen wir lieber noch ein Beispiel eines harmlosen Zeigers. Diesen Zeiger liefert das BASIC auf Anfrage, und er gibt die Position einer Variablen im Speicher an. Gemeint ist die VARPTR()-Funktion. Mit Kenntnis dieser Adresse können wir z.B. Texte manipulieren. Dabei müssen Sie allerdings beachten, daß die VARPTR()-Funktion sich bei der 138-KByte-BASIC-Version von den anderen Versionen unterscheidet. Sollten Sie diese bestimmte Version des BASIC besitzen, müssen Sie anstatt der Befehlsfolge "VARPTR(a$)+2" immer den Befehl "VARPTR(a$)" verwenden.

```
10   rem *** VARPTR()-Demonstration ***
20   a$ = " Nanu !"            : rem Textvariable definieren
30   a = peek(varptr(a$)+2)    : rem Speicherzellen ermitteln
40   print a$                  : rem Vorher...
50   poke a,65 : print a$      : rem und Nachher !
```

Aus "Nanu" wird "Au", was ohne direkte Manipulation der Variable A$ vor sich ging. Durch die Verwendung von Vektoren sind die Anwendungsmöglichkeiten der Peek- und Poke-Befehle also noch wesentlich erweitert!

Hier noch ein kleines Programm, mit dem man "den Ritt durchs wilde Pokeistan" vornehmen kann. Nach Eingabe der Speicheradresse (0 erwirkt Abbruch) gibt es den momentanen Inhalt dieses Wortes an und fragt nach dem neuen Wert. Will man ihn unverändert lassen, so betätigen Sie einfach die Return-Taste und die nächste Adresse wird abgefragt.

```
10  rem *** PEEK- und POKE- Anwendung ***
20  input "Adresse ";a          : rem Eingabe
30  if a=0 then end             : rem Abbruch bei 0
40  print peek(a); " => ";      : rem alter Wert
50  input x$                    : rem neuer Wert
60  if len(x$) = 0  then 20 : rem keine Änderung
70  poke a,val(x$) : goto 20 : rem und weiter
```

Dieses Programm sollte man am besten abspeichern, da man damit wahrscheinlich noch oft herumpoken wird...

2.1.5 Datenstapel: Stacks

Stellen Sie sich vor, Sie würden ein Maschinenspracheprogramm schreiben und müßten ein paar Daten irgendwo ablegen. Die Frage ist, wohin man sie legen kann, ohne daß sie etwas wichtiges überschreiben oder von einem anderen Programm geändert werden. Dafür haben sich die Konstrukteure von Microprozessoren etwas ganz besonderes einfallen lassen. Wahrscheinlich haben sie auf ihren Schreibtisch geschaut und sich gesagt, wenn ich einen Zettel auf einen Haufen lege - warum soll ein Computer das nicht auch können?

Aus dieser Idee ist das Prinzip des Stacks entstanden, was ja nichts anderes als Stapel heißt. Ein Prozessor bekommt einen besonderen Speicherbereich zugeordnet, wo er nach Belieben seine Zettel (sprich Daten) drauflegen und wieder herunternehmen kann. Der im ATARI ST eingebaute MC 68000 hat dergleichen sogar zwei, je einen für den Überwacher, auch Supervisor genannt, und einen für den User. Es ist bei diesem Prozessor sogar möglich, noch mehr Stacks für eigene Zwecke

anzulegen, die von selbstgeschriebenen Programmen verwendet werden können.

Bei diesen Stapeln handelt es sich, wie auch beim Zettelkasten des imaginären Prozessorbauers, um einen LIFO-Stack. LIFO bedeutet Last In-First Out, also werden die Daten, die zuletzt auf den Haufen gelegt wurden, auch zuerst wieder heruntergenommen. Das Gegenstück dazu sind FIFO-Stacks (First In-First Out), die bei anderen Vorgängen eine große Rolle spielen. Bleiben wir beim LIFO-Stack des Prozessors. Hier werden außer Daten noch Adressen abgelegt. Dies ist eigentlich sogar die Hauptanwendung eines Stacks. Wenn der Prozessor in eine Unterroutine springt, was etwa dem BASIC-Befehl GOSUB entspricht, legt er die Adresse, von der er herkommt, im Stack ab. Beim RTS, entsprechend dem RETURN in BASIC, nimmt er diese Adresse zurück und arbeitet ab da weiter. Dieser Ablauf gilt auch für BASIC-Unterprogramme.

Auf diese Weise können wir innerhalb eines Programmes leicht vorübergehend Daten ablegen, die wir im Moment nicht brauchen. Bei größerem Datenumfang muß allerdings ein eigener Speicherbereich verwendet werden. Hat man jedoch keinen Platz im Arbeitsspeicher des Computers, muß man seine Daten eben außerhalb des Rechners ablegen. Hierfür eignen sich besonders gut Disketten, deren Funktion wir nun betrachten wollen.

2.2 Disketten

Eine sehr oft angewandte Methode der Datensicherung ist die Speicherung auf Diskette. Das ATARI-Laufwerk verwendet hierzu 3½ Zoll-Disketten, die zwar (noch) etwas teuer sind, aber dafür recht handlich und sicher. Momentan am weitesten verbreitet ist das 5¼ Zoll-Format, bei größeren Anlagen findet man auch 8 Zoll-Laufwerke. Die Technik der Abspeicherung ist jedoch bei allen Formaten gleich.

Wenn Sie die Schutzblende einer einer 3½ Zoll-Diskette zur Seite schieben, sehen Sie eine dunkelbraune Scheibe mit dem Durchmesser von eben 3½ Zoll. Diese Scheibe ist wie ein Tonband mit

einem magnetisierbaren Material beschichtet und wird im Laufwerk mit konstanter Geschwindigkeit gedreht. Auf der Scheibe liegt dann der Schreib-/Lesekopf und erfaßt die magnetischen Daten. Beim einseitigen Laufwerk wird nur die Unterseite der Diskette benutzt. Die obere Beschichtung der Floppy ist deshalb bei Single-Sided-Disketten nicht getestet, weshalb ihre Verwendung in zweiseitigen Laufwerken das Risiko birgt, durch sogenannte "Dropouts" Daten zu verlieren.

Eingeteilt ist eine Diskette in Spuren und Sektoren. Jede Spur, auch Track genannt, enthält 9 Sektoren mit je 512 Bytes. Das ATARI SF 354-Laufwerk verwaltet 80 Tracks auf einer Diskette, wodurch sich eine Gesamt-Speicherkapazität von 512 * 9 * 80 = 368640 Bytes ergibt. Der Floppy-Controller im ATARI ST liefert nun auf Anfrage einen Sektor oder einen ganzen Track. Der Anwender braucht aber nicht zu wissen, in welchen Sektoren nun sein Programm liegt, diese Informationen sucht sich das Betriebssystem selbst. Es findet sie im Inhaltsverzeichnis der Diskette, wo außer Name, Länge und Datum des Programmes noch Position und Status (Lesen und Schreiben oder nur Lesen, siehe Datei-Info im Menü) verzeichnet ist.

Dieses komplette Inhaltsverzeichnis liegt im Speicher, wo sich beim Aufruf eines Programmes das TOS die benötigten Parameter holt. Dies birgt allerdings eine Gefahr: nicht immer merkt der Rechner, wenn Sie die Diskette gewechselt haben. Klicken Sie also ein Programm aus dem alten Inhaltsverzeichnis an, obwohl Sie die Diskette gewechselt haben, so lädt das Betriebssystem die Sektoren ein, in denen das Programm eigentlich stehen müßte. Dies kann dann durch die meist total falschen Daten zu einem Absturz führen! Betätigen Sie daher nach jedem Diskettenwechsel die Escape-Taste, um das aktuelle Fenster zu aktualisieren.

2.2.1 Daten-Zwischenspeicherung

Wollen Sie nun eine Diskette zum Abspeichern von irgendwelchen Daten (z.B. Text) verwenden, so müssen Sie zuerst eine Datei eröffnen. Das BASIC bietet hierfür den OPEN-Befehl an. Zur Demonstration einer solchen Dateiverwaltung geben Sie folgenden Programmteil ein:

```
5     input "Dateiname ";f$          : rem z.B. TEST.TXT
10    input "(1) Eingabe / (2) Ausgabe / (3) Ende" ;a
20    on a goto 50,100
30    end
40    rem *** Texteingabe ***
50    print "Eingabe :  ('x'=Abbruch)"
60    open "O", #1, f$, 128          : rem O für Output
70    input a$
80    print#1,a$ : if a$="x" then close #1 : goto 10
90    goto 70
95    rem *** Textausgabe ***
100   print "Ausgabe :"
110   open "I", #1, f$, 128          : rem I für Input
120   input#1,a$ : if a$="x" then close #1 : goto 10
130   print a$ : goto 120
```

Dieses Programm zeigt die Verwendung der OPEN-, INPUT#- und PRINT#-Befehle. Die Datenrichtung (von oder zur Diskette) muß bereits im OPEN-Befehl definiert werden. Mit der PRINT#-Anweisung können Sie natürlich beliebige Texte oder auch Zahlenwerte aus numerischen Variablen übergeben und entsprechend über Input# wieder laden.

2.2.2 Graphik ablegen

Eine weitere Möglichkeit der Datenspeicherung bieten die BLOAD- und BSAVE-Anweisungen. Mit ihnen kann man Speicherbereiche direkt von der Diskette laden bzw. auf ihr ablegen. So können Sie z.B. selbstgeschriebene Maschinenspracheprogramme an eine definierte Speicherstelle laden und mit CALL aufrufen. Sie können damit auch den Inhalt des Bild-

schirmspeichers abspeichern, wenn Sie eine Grafik erstellt haben. Dies geht folgendermaßen vor sich:

```
110 rem *** Bild abspeichern ***
120 defdbl a : a=1102
130 b=peek(a)
140 bsave "bild.dat", b, 10000
150 return
200 rem *** Bild laden ***
200 defdbl a : a=1102
200 b=peek(a)
200 bload "bild.dat", b
210 return
```

Die beiden Teile des Programmes werden als Unterroutinen mit GOSUB 100 bzw. GOSUB 200 aufgerufen. Der gewählte Dateiname "BILD.DAT" ist dabei willkürlich gewählt und kann beliebig geändert werden. Allerdings darf der Dateiname höchstens 8 Buchstaben lang sein und einen dreistelligen Typen (hier .DAT) besitzen, wobei der Punkt nicht zählt. Darüberhinaus gehende Zeichen werden ignoriert. Des weiteren ist zu beachten, daß das erste Zeichen im Namen ein Buchstabe (A-Z) ist.

3. Computer-Einmaleins

Bei der Arbeit mit Computern, besonders mit der Maschinensprache und den verwandten BASIC-Befehlen PEEK und POKE, bekommt man es immer wieder mit verschiedenen Darstellungsarten von Zahlen zu tun. Aus diesem Grund wollen wir nun einen kleinen Ausflug in die Welt der Zahlensysteme unternehmen. Wenn Sie bereits mit dieser Materie vertraut sind, dann können Sie getrost weiterblättern, da Sie hier wahrscheinlich nichts Neues finden.

Alle Adressen in den bisher aufgeführten Tabellen sind in Sedezimal, auch Hexadezimal genannt, angegeben. Normalerweise werden sie entweder mit einem $-Zeichen vor oder einem "H" nach der Zahl gekennzeichnet. Es handelt sich dabei um ein Zahlensystem, welches sich deutlich von dem wohlbekannten Dezimalsystem unterscheidet, da hier nicht nur die Ziffern 0 bis 9, sondern auch die Buchstaben A bis F auftreten. Der Grund, weswegen man diese etwas ungewohnte Schreibweise bevorzugt, liegt einerseits in der kleineren Stellenzahl, die zum Ausdruck dieser hohen Werte benötigt wird ($FC 0000 entspricht 16.515.072 Dezimal).

Andererseits, und das ist der entscheidende Vorteil des Sedezimalsystems, beschreibt es die Speicheraufteilung eines Digitalrechners wesentlich übersichtlicher. So kann ein Byte die Werte zwischen 0 und 255 annehmen, also 256 verschiedene Zustände. Nimmt man nun nicht 10 als Zahlenbasis wie im Dezimalsystem (Dezi = zehn), sondern 16, so kann man mit zwei Ziffern genau diese 256 bzw. $FF Zustände beschreiben. Der Zeichenvorrat wird hier von 0..9 auf 0..F erweitert, wobei ein A der 10, ein B der 11 usw. entspricht.

Der Prozessor, der mit diesen Zahlen arbeiten muß, verwendet aber weder das Dezimal- noch das Sedezimalsystem. Für ihn gibt es nur zwei mögliche Zustände, 5 Volt oder 0 Volt. Nur das kann er verstehen und direkt verarbeiten. Man nennt diesen Blickwinkel auch die Maschinenebene.

Hier bestehen alle Daten und Adressen aus einer Folge von Einsen und Nullen. Diese Technik der Kodierung mittels zweier Zustände nennt man digital (di = zwei). Nehmen wir z.B. die Zahl 100. Diese würde abgespeichert in einem Byte als "01100100", wobei jedes Bit eine Zweierpotenz vertritt. Das rechte Bit hat also den Wert 2^0 (1), das nächste 2^1 (2), dann 2^2 (4) usw. Diese Werte werden, wenn das entsprechende Bit den Wert 1 hat, addiert. In unserem Beispiel hieße das:

$$2^2 + 2^5 + 2^6 = 4 + 32 + 64 = 100.$$

Diese Methode ist allerdings etwas mühsam, denke man nur an die 24 Bit breiten Adressen im ATARI! Dennoch hat es viele Vorteile, wenn man die verschiedenen Zahlensysteme kennt. Auch sollten wir Zahlen von einem ins andere System umwandeln können.

3.1 Zahlensystem-Umwandlung

Diese Umrechnung ist zugegebenermaßen umständlich. Doch wozu besitzt man einen Computer? Schreiben wir doch ein Programm in BASIC, welches das auch kann! Hier sind nun Programmvorschläge, die diese Umrechnungen vornehmen. Leider mußte ich einige Klimmzüge machen, da die Rechengenauigkeit meiner vorliegenden BASIC-Version etwas zu wünschen übrig ließ. Außerdem sei noch auf den HEX$-Befehl hingewiesen, der aber nur in Verbindung mit einem direkten numerischen Wert, z.B. PRINT HEX$(100), oder mit einer Integer-Variablen, z.B. PRINT HEX$(X%), arbeiten und daher nur Sedezimalzahlen bis zu $FFFF ausgeben kann.

```
10  defdbl e,d                  : rem Variablen-Definitionen
20  defint n
30  z$= "0123456789ABCDEF"
40  input "Dezimalzahl ";d  : rem Dezimal => Hexadezimal
50  for i = 5 to 0 step -1 :e = 16^i
60  n = .1 + d / e : d = d - n * e
70  print mid$(z$,n+1,1);
80  next i :? :? h$ :end
```

```
90   defdbl e,d                 : rem Variablen-Definitionen
100  defint n
110  z$= "0123456789ABCDEF"
120  input "Hexzahl ";h$        : rem  Hexadezimal => Dezimal
130  d=0 : for i = 1 to len(h$)
140  e = 16^ (len(h$)-i)
150  d=d + e * (instr(1,z$,mid$(h$,i,1))-1)
160  next i : print d : end

170  input "Dezimalzahl ";d     : rem  Dezimal => Binär
180  print d;" => "; : for i = 15 to 0 step -1
190  if (d and 2^i) then ? "1";  else print "0";
200  next i : print : end

210  input "Binärzahl ";b$      : rem  Binär => Dezimal
220  print b$;" = "; : d = 0
230  for i = 1 to len(b$) : e = len(b$)-i
240  d= d - (mid$(b$,i,1) = "1") * 2^e
250  next i : print d : end
```

3.2 Bit-Auswertung (Beispiel: Joystick)

Gerade die Binärdarstellung von Zahlen spielt bei der Speicherplatz-Manipulation eine große Rolle. Oft sind nämlich Funktionen abhängig von einem bestimmten Bit, wie z.B. die Auswertung einer Joystickstellung. Die Zahl, die man bei der Abfrage des Joysticks erhält, gibt in vier Bits die Zustände der vier Schalter an, die beim Bewegen des Knüppels betätigt werden. Bit 0 und 1 stehen für die vertikale Bewegung des Joysticks, Bit 2 und 3 für die horizontale Richtung. Hierzu ein kleines Beispielprogramm:

```
10   rem *** Joystick - Auswertung ***
20   out 4,22                : rem Anweisung an die Tastatur
30   defseg =1
40   js = peek(3591)         : rem Ergebnis holen
50   print js,               : rem Gesamtwert ausgeben
```

```
60   Y = Y + (js and 2)/2 - (js and 1)
70   X = X + (js and 8)/8 - (js and 4)/4
80   print x,y              : rem Koordinaten ausgeben
90   goto 20                : rem Endlos-Schleife
```

Dieses Programm fragt den Joystick 1 ab und verändert eine mit X und Y angegebene Position (z.B. Koordinaten eines Cursors) in Abhängigkeit von dessen Stellung. Hier werden mit dem AND-Befehl die jeweiligen Bits getestet. Diese AND-Verknüpfung ist eng verwandt mit den BASIC-Funktionen OR, NOT und XOR. Diese Verknüpfungen werden binär vorgenommen, indem die beiden Werte Bit für Bit verglichen werden. Was bei diesem Vergleich schließlich herauskommt, zeigt das folgende Kapitel.

3.3 Logische Operationen

0 AND 0 = 0	
0 AND 1 = 0	
1 AND 0 = 0	Das Bit im Ergebnis wird nur gesetzt,
1 AND 1 = 1	wenn beide getesteten Bits 1 sind.

0 OR 0 = 0	
0 OR 1 = 1	Das Bit im Ergebnis wird immer dann
1 OR 0 = 1	gesetzt, wenn eines der getesteten
1 OR 1 = 1	Bits 1 ist.

0 XOR 0 = 0	
0 XOR 1 = 1	Das Ergebnisbit wird immer dann zu 1,
1 XOR 0 = 1	wenn die getesteten Bit unterschiedlich
1 XOR 1 = 0	sind.

Schließlich die NOT-Operation, die jedes Bit ins Gegenteil verkehrt, also

NOT 0 = 1
NOT 1 = 0

Das Beispielprogramm der Joystickauswertung testet mit der AND-Verknüpfungden Wert von JS. Ist dieser Wert = 3 und soll das 1. Bit getestet werden, läuft die Operation JS AND 1 so ab:

JS = 3, das entspricht	binär	0011
Testbyte = 1,	binär	0001
Ergebnis	3 AND 1:	0001

Das Ergebnis ist also immer dann ungleich 0, wenn das getestete Bit 1 ist. Auf diese Weise kann mit vier solcher Vergleiche der Joystick ausgewertet werden.

Eine weitere Anwendung der logischen Operationen ist die gezielte Manipulation eines Bits. Hat man ein Datenbyte vorliegen, aus dem ein bestimmtes Bit gelöscht werden soll, so verwendet man auch die AND-Verknüpfung.

Beispiel:

Datenbyte ist		00110011
Bit 4 ist zu löschen, also	AND	11101111
ergibt (Bit gelöscht!)		00100011

Diesen Vorgang nennt man Maskieren, wobei das 11101111-Wort die Maske darstellt. Eine ähnliche Methode wird beim setzen eines Bits verwendet. Hier nimmt man die OR-Verknüpfung.

Beispiel:

Datenbyte ist		00110011
Bit 3 ist zu setzen, also	OR	00001000
ergibt (Bit gesetzt!)		00111011

Die XOR-Verknüpfung bietet eine weitere interessante Möglichkeit. Mit ihr können Bits umgeschaltet werden, d.h. ein gesetztes Bit wird gelöscht und umgekehrt. Wird die gleiche Operation mehrmals durchgeführt, nimmt das Bit immer abwechselnd die Zustände 0 und 1 an.

Beispiel:

Datenbyte ist		00110011
Bit 0 wird umgeschaltet	XOR	00000001
ergibt		00110010
und nochmal	XOR	00000001
ergibt wieder den		
ursprünglichen Wert!		00110011

Und schließlich ist noch die NOT-Operation zu nennen, die eine gewisse Ähnlichkeit zu XOR besitzt. NOT invertiert alle Bits in einem Datenbyte, so daß sich ein Wert ergibt, der dezimal 255 minus dem ursprünglichen Byte entspricht.

Beispiel:

Datenbyte ist		00110011	= 51
	NOT	00110011	
ergibt		11001100	= 204 + 51 = 255

Dies waren die Binär-Operationen, die das BASIC mit entsprechenden Befehlen unterstützt. Arbeitet man jedoch in Maschinensprache, so gibt es noch einige mehr. So kann man ein Datenwort links- oder rechtsherum verschieben bzw. rollen. Der Effekt dabei ist der, daß sich der Wert des Wortes verdoppelt bzw. halbiert. Ein Beispiel:

Datenbyte ist	00110011	= 51
linksverschoben:	01100110	= 102, also 51*2
rechtsverschoben:	00011001	= 25, also 51/2 (integer)

Das Byte wird beim Verschieben mit Nullen aufgefüllt. Zusammengefaßt heißt das, daß n mal nach links verschoben der Multiplikation mit 2^n entspricht, n mal nach rechts bedeutet die Division durch 2^n.

Ein weiterer Punkt in der Binärarithmetik ist die binäre Darstellung von negativen Zahlen. Hier wird immer das höchstwertigste Bit des Datenwortes als Vorzeichen gesehen. Ist dieses Bit = 1, so handelt es sich in manchem Zusammenhang um eine

negative Zahl. Die Beachtung des Vorzeichens ist nicht immer nötig, da z.B. Adressen nur einen absoluten Wert besitzen können. Beim PEEK-Befehl findet man dies jedoch vor, da (in Wortdarstellung) alle Werte über 32767 als negativ dargestellt werden. Nach einem POKE x,65535 erhält man somit durch PEEK(x) den Wert -1, da die verbleibenden Bits als Zahl 32768 minus diese Zahl interpretiert werden.

Diesen Zusammenhang zwischen einer Zahl und ihrer negativen Darstellung nennt man ein Zweierkomplement. Der NEG-Befehl der MC 68000-Maschinensprache bildet ein solches Komplement, indem er den zu negierenden Wert von 0 subtrahiert.

Eine andere Darstellung von Zahlen in der Binärebene ist die BCD-Codierung. BCD bedeutet Binary Coded Decimal und stellt eine dezimal orientierte Kodierung dar. Hier wird das Binärwort in Blöcke zu je 4 Bit aufgeteilt, in denen jeweils nur die Werte 0 bis 9 enthalten sind. Somit wird ein 16 Bit-Wort nicht vollständig ausgenutzt, da es in BCD-Darstellung maximal den Wert 9999 annehmen kann, während in normaler Kodierung 65535 erreicht wird.

Der Vorteil dieser Kodierung liegt aber in der einfacheren Darstellung und Verarbeitung von Dezimalzahlen. Die Umwandlung einer BCD-kodierten Dezimalzahl ist dabei genauso einfach, wie man aus einer normalen Binärzahl den hexadezimalen Wert erhalten kann. Man braucht ja nur Bitblock für Bitblock in eine Ziffer umzuwandeln und erhält so direkt den kodierten Wert. Ein Beispiel für die BCD-Kodierung finden Sie in dem Auswertungsprogramm für die Uhrzeit in dem Kapitel über die intelligente Tastatur. Diese Uhrzeit, wie auch das Datum, wird dabei nämlich auch BCD-mäßig im Speicher abgelegt.

[illegible] zur Beachtung des Vorzeichens ist nicht immer [illegible] Adressen nur einen absoluten Wert werfen [illegible] PEEK-Befehl findet man dies jedoch vor, da [illegible] alle Werte über 32767 als negativ [illegible] einem [illegible] 65535 erhalten [illegible] somit [illegible] da [illegible] vorstehenden [illegible] die Zahl [illegible] interpretiert werden.

[illegible] zwischen einer Zahl und ihrer [illegible] ein Zweierkomplement [illegible] bringt ein solches [illegible] Wert von 0 subtrahiert.

[illegible] von Zahlen in der [illegible] Binary Coded Decimal und [illegible] Codierung dar. Hier wird [illegible] aufgeteilt, in [illegible] Sonst wird ein 16 Bit-Wort nicht [illegible] BCD-Darstellung maximal den [illegible]

[illegible] Kodierung liegt aber in der einfachen [illegible] von Dezimalzahlen. Die Umwandlung [illegible] normalen [illegible] nur Bitblock [illegible] direkt den kodierten [illegible] BCD-Kodierung finden Sie in dem [illegible] die Uhrzeit [illegible] das Datum [illegible] im Speicher abgelegt.

4. Die programmierte Persönlichkeit

Ein Computer ist im Moment des Einschaltens wirklich unglaublich dumm. Hat er auch noch so viele Fähigkeiten, so weiß er doch damit nichts anzufangen. Er braucht ein Programm, das ihm sagt, was er zu tun hat. Ein solches Programm, das das gesamte Computersystem erst ausmacht, nennt sich Betriebssystem. In der Vergangenheit haben sich einige dieser Systemprogramme bewährt, wie z.B. CP/M, welches mit dem Prozessor Z80 A arbeitet, oder MS-DOS, das schon mit quasi-16-Bit-Prozessoren wie dem 8088 im IBM-PC arbeiten kann. In Verbindung mit dem MC 68000 gibt es das CP/M 68K, welches ursprünglich auch in den ATARI ST eingebaut werden sollte. Doch dieses Betriebssystem hat einige Nachteile, wie zum Beispiel die nicht gerade umwerfende Geschwindigkeit, und ist natürlich auch nicht in der Lage, die umfassenden Fähigkeiten des ST zu bedienen. So entwickelte das Softwarehaus Digital Research, aus deren Feder auch CP/M stammt, das speziell auf die ST-Serie abgestimmte GEMDOS. Dieses Betriebssystem ist auch als TOS bekannt geworden.

4.1 Das Tramiel-Operating-System

Diese Bezeichnung des GEMDOS ist nach dem "Vater" des ATARI ST benannt und beinhaltet nach von CP/M her gewohnter Manier Disketten- und Peripherieverwaltung. Doch das TOS kann noch wesentlich mehr. Um den vielseitigen Fähigkeiten der ATARI ST-Serie gerecht zu werden, wäre die Verwendung von CP/M mit zu vielen Abstrichen verbunden. CP/M 68K, wie die Version dieses Betriebssystems für den MC 68000-Prozessor heißt, hat außerdem den Nachteil, nicht gerade schnell zu arbeiten. Aus diesen Gründen hat ATARI sich entschlossen, ein völlig neues Betriebssystem für das neue Flaggschiff ST zu entwickeln. Einige Ideen des CP/M 68K sind in GEMDOS übernommen worden, jedoch sind noch mehr Spezialitäten eingebaut. Die von MS-DOS (ein IBM-Betriebssystem) her bekannte hierarchische Struktur des Inhaltsverzeichnisses ist auch im ST realisiert. Dieser Struktur verdankt der ATARI ST-

Besitzer die Möglichkeit, Unterverzeichnisse in Form von Ordnern anzulegen.

4.1.1 Zusammensetzung

Ein wichtiger Teil des TOS ist das BIOS (Basic Input Output System), das die Verbindung zwischen der Soft- und der Hardware bildet. Durch eine relativ einfache Befehlsstruktur ist das BIOS für den Programmierer eine enorme Erleichterung. So braucht er sich z.B. nicht über die Signalfolge der seriellen Schnittstelle Gedanken zu machen, sondern er gibt dem BIOS einfach einen Befehl zur Ein-/Ausgabe. Die Art und Weise der BIOS-Programmierung ist jedoch nur für den interessant, der mit Programmiersprache C oder Maschinensprache programmiert. BASIC-Benutzer haben es da einfacher, da Sie beispielsweise nur PRINT eingeben und der BASIC-Interpreter den Rest erledigt.

Doch ist dieses alte BIOS nicht ausreichend für einen Rechner vom Kaliber der ST-Serie. Deshalb wurde das BIOS mit einigen zusätzlichen Befehlen erweitert. Dazu gehören Bildschirm-, Disketten- und MIDI-Operationen ebenso wie die eingebaute Hardcopy-Funktion. Eine Auflistung der immerhin 52 Befehle von BIOS und extended BIOS (XBIOS) ist aber nicht nötig, da spezielle Befehlsanfrage , -übergabe oder -übernahme der Parameter nur den maschinennahen Programmierern möglich ist (siehe Abschnitt 6.3 oder 6.4). Die BASIC-Fans sparen hier einiges Tabellenwälzen. Allerdings gibt es Funktionen des BIOS, die das BASIC nicht beherrscht. So ist es nicht möglich, die Drucker- oder Modem-Schnittstelle zu konfigurieren (nur vom DESKTOP aus). Aber diese Nachteile kann man in Kauf nehmen, wenn man den Komfort einer Hochsprache wie BASIC bedenkt. Aber eine Hintertür hält diese Sprache offen: die PEEK- und POKE-Befehle. Hiermit lassen sich auch einige Systemparameter modifizieren, deren Zugriff eigentlich dem TOS vorbehalten sind.

Das TOS wird in der vorliegenden Version von der Diskette geladen, wenn der Rechner eingeschaltet wird. Es belegt dann den Speicherbereich von $500 bis ca. $32000 und beinhaltet direkt das Graphikpaket GEM und die Resource-Parameter, die die Menüs und die Alarmfenster ausmachen. Dieser riesige Speicherplatz wird allerdings nicht nur von den Programmen belegt, es befinden sich hier auch etliche Angaben über den Zustand des Systems beim Start. Ist das TOS vollständig geladen, so tritt das BIOS in Aktion (ab $6100) und versetzt den gesamten Rechner in seinen Grundzustand. Danach sucht es auf der Diskette nach Programmen mit der Kennung .ACC und lädt diese mit in den Speicher. Diese sogenannten Accessory-Programme sind dann ständig in Bereitschaft und können vom Benutzer über den Menüpunkt DESK aufgerufen werden. Die auf der gelieferten Diskette vorhanden zwei ACC-Dateien bilden das Kontrollfeld, die Schnittstellenanpassungen und den Terminal-Emulator. Wenn Sie diese Funktionen nicht benötigen, können Sie durch Umbenennung der Programme (z.B. auf DESK1.AC) deren Speicherplatz sparen. Sie gewinnen dadurch etwa 20 KByte für Ihr BASIC-Programm!

4.1.2 Systemvariablen

Noch einen Speicherbereich initialisiert das BIOS. Es handelt sich hier um eine Tabelle von Systemparametern, die teilweise geändert werden dürfen. Auch enthält dieser Bereich viele Vektoren, die auf Interruptprogramme zeigen, das sind Maschinenprogramme zur Behandlung von Programmunterbrechungen. Diese Vektoren sollten keinesfalls "verbogen" werden, da eine Änderung höchstwahrscheinlich einen Systemabsturz verursacht (und schon wieder Bömbchen auf dem Bildschirm...). Die Tabelle liegt zwischen $400 und $4FF (1024 - 1279) und beinhaltet folgendes:

$400 - $41F	Sprungadressen (Vektoren)
$420	enthält eine "magische Zahl", die einen erfolgreichen Kaltstart anzeigt
$424	Speicher-Konfiguration (= 4 bei 512 KByte, 5 bei 1 MByte)

$426 noch eine "magische Zahl", die bewirkt, daß bei Reset ein Kaltstart erfolgt
$42A Zeiger auf die Kaltstart-Routine
$42E Zeiger auf RAM-Ende ($80000 bei 512 KByte $F0000 bei 1 MByte)
$432 Zeiger auf Arbeitsspeicher-Anfang
$436 Zeiger auf Arbeitsspeicher-Ende
$43A schon wieder eine magische Zahl, die einen erfolgreichen Kaltstart anzeigt
$43E DMA-Flag, muß <>0 sein!
$440 eingestellte Floppy-Geschwindigkeit (3)
$442 System-Timer in Millisekunden (20 für 50 Hz)
$444 Floppy-Vergleichs-Flag. Wenn <>0, dann wird jeder Schreibvorgang durch Lesen getestet. Wenn 0, dann nicht (schreibt schneller)
$446 Gerätenummer, von dem das System geladen ist
$448 PAL/NTSC-Flag (<>0: PAL; =0: NTSC)
$44A eingestellte Bildschirmauflösung
$44C aktuelle Bildschirmauflösung (0-2)
$44E Zeiger auf Bildschirmspeicher-Anfang
$452 VBI-Flag. Sollte 1 sein (VBI = Vertical Blank Interrupt)
$454 VBI-Routinenanzahl (8)
$456 Zeiger auf VBI-Zeiger-Tabelle
$45A Zeiger auf neue Farb-Tabelle (wenn neu)
$45E Zeiger auf neuen Bildspeicher (wenn neu)
$462 VBI-Zähler
$46A ausgeführte VBI Zähler
$46E Zeiger auf Routine bei Monitorwechsel (Kaltstart)
$472-$481 Festplattenparameter (0 wenn keine vorhanden)
$482 wenn <>0, ist COMMAND.PRG geladen worden
$484 Tastatur-Status (Bit 0: Glocke an/aus; Bit 1: Keyrepeat an/aus; Bit 2: Tastenklick an/aus)
$48E Arbeitsspeichergrenzen (nicht verändern!)
$4A2 Zeiger auf Registerspeicher des BIOS
$4A6 Anzahl der angeschlossenen Disk-Laufwerke
$4AE Zeiger auf Prozessorzustands-Speicher
$4B4 Zeiger auf Datensektor-Zwischenspeicher
$4B8 Zeiger auf Inhaltsverzeichnis-Puffer
$4BC 200 Hz-Zähler

$4C4	ist 3, wenn Floppies angeschlossen sind
$4C6	Zeiger auf 1 KByte Disk-Puffer
$4EE	Hardcopy-Flag (wenn 0, dann Bildausdruck)
$4F2	Zeiger auf Betriebssystem-Start
$4F6	Zeiger auf Graphik- bzw. Text-Segment
$4FA	Zeiger auf Betriebssystem-Ende
$4FE	Zeiger auf AES-Textsegment

4.1.3 Der TOS-Bildschirm

Die Benutzerebene des GEM-Desktop ist nicht die einzige, über die der ATARI ST verfügt. Es existiert noch eine andere, die allerdings nur für Text geeignet ist. Auf diesem "Blatt Papier" arbeiten alle Programme, die als Typ die Bezeichnung .TOS tragen. Ebenfalls verwendet wird es in der VT52-Emulation, die unter dem Menüpunkt DESK zu finden ist. In dieser Ebene funktioniert die gesamte Cursorführung mit Steuerzeichen, die mit einem Druck auf die Escape-Taste begonnen werden. In folgender Tabelle finden Sie diese Steuersequenzen. Esc steht für Escape.

Esc A	Cursor hoch
Esc B	Cursor runter
Esc C	Cursor rechts
Esc D	Cursor links
Esc E	Bild löschen und Cursor nach links oben
Esc H	Cursor in linke obere Ecke
Esc I	Cursor hoch, scrollen wenn nötig
Esc J	Bildschirm ab Cursor löschen
Esc K	Zeile ab Cursor löschen
Esc L	Zeile einfügen
Esc M	Zeile löschen
Esc Y (y+32)(x+32)	Cursor in X/Y-Position bringen
Esc b Farbe	Schriftfarbe wählen (0-16 bei Farbbild)
Esc c Farbe	Hintergrundfarbe einstellen (0-16)
Esc d	Bildschirm bis Cursor löschen
Esc e	Cursor einschalten
Esc f	Cursor ausschalten

Esc j	Cursorposition speichern
Esc k	Cursor auf gespeicherte Position setzen
Esc l	Zeile löschen
Esc o	Zeile bis Cursor löschen
Esc p	Reversdarstellung einschalten
Esc q	Reversdarstellung ausschalten
Esc v	Überlauf an
Esc w	Überlauf aus (Cursor bleibt am rechten Rand stehen)

Diese Funktionen sind wie gesagt im VT52-Emulator gültig. Man kann sie aber auch in einem BASIC-Programm verwenden. Somit ist es möglich, die OUTPUT-Fenster-Umrahmung zu löschen und den gesamten Bildschirm für das Programm zu nutzen. Doch Vorsicht, eine gelöschte Menüzeile bleibt gelöscht, auch wenn sie noch funktioniert. Man muß sie nach Rückkehr in die normale BASIC-Ebene neu schreiben.

Mit dem folgenden Programm kann man das alles ausprobieren:

```
10   rem *** TOS - Ebene  Demo ***
20   x = inp(2)           : rem Tasten holen
30   if x = 187 then end : rem F1 bewirkt Abbruch
40   out 2,x              : rem Ausgabe an TOS-Bildschirm
50   goto 20              : rem Endlos-Schleife
```

Diesen Trick kann man in einem BASIC-Programm auch dafür einsetzen, um zwei voneinander unabhängige Cursors zu bekommen. Um vom BASIC-Programm aus den TOS-Cursor zu steuern bzw. an dieser Stelle etwas auszugeben, kann dieses Programm verwendet werden:

```
10   rem *** Ausgabe an den TOS-Cursor ***
20   esc$=chr$(27)                : rem Escape definieren
30   aus$ = esc$+"B"+esc$+"b2"    : rem Cursor runter und Farbe
40   aus$ = aus$+"Hallo !"        : rem Text
50   for i= 1 to len(aus$)        : rem Ausgabeschleife
60   out 2,asc(mid$(aus$,i,1))    : rem Zeichen übergeben
70   next i                       : rem bis fertig
```

Die Zusammensetzung der AUS$-Variablen kann auch auf eine andere Art geschehen, die Ausgabeschleife gibt den gesamten Text inclusive Steuerzeichen an das TOS weiter. Es können dabei also alle oben aufgeführten Steuerzeichen verwendet werden. Übrigens: sollte Ihnen die Blinkfrequenz des TOS-Cursors nicht zusagen, können Sie diese mit POKE 10530,n nach Ihrem Belieben ändern!

Dabei ergibt sich eine weitere Möglichkeit, in einem Programm eine definierte Zeit festzustellen. Die in der Speicherzelle 10530 eingestellte Zahl gibt nämlich den Wert an, der nach jeder Änderung des Cursorzustandes in die nächste Adresse, 10531, geladen wird, die dann auf Null heruntergezählt wird. Dieser Countdown erfolgt automatisch, so daß auf jeden Fall irgendwann der Wert 0 wieder erreicht wird und der Cursor wieder seinen Zustand ändert. Man kann diesen Effekt auch so verwenden, daß man die Verzögerung in 10530 auf einen bestimmten Wert setzt und z.B. innerhalb einer Schleife wartet, bis in 10531 wieder eine Null steht. In einer solchen Schleife kann dabei auch auf ein anderes Ereignis gewartet werden. So kann ein Programm auf eine Eingabe warten und dem Benutzer nur soundsoviel Zeit dabei lassen (Reaktionstest).

Wenn wir gerade bei Programmverzögerungen und Warteschleifen sind, sollte noch erwähnt werden, daß mit Hilfe des SOUND-Befehls ebenfalls eine definierte Zeit gewartet werden kann. Dabei braucht man nur den Ton selbst unhörbar einzustellen und dabei die Tondauer als Verzögerungswert einzustellen. Die Dauer wird dabei in 1/50 Sekunden angegeben. Der Befehl SOUND 1,0,1,1,50 läßt das Programm zum Beispiel 1 Sekunde anhalten.

4.2 GEM

GEM, der Graphics Environment Manager, wurde von Digital Research als neue Benutzerschnittstelle entwickelt. Es erinnert an die Bedieneroberfläche des Mackintosh von APPLE, der diese Art der graphisch orientierten Bedienung eines Computers erstmals demonstrierte. Hier wird die sonst übliche Methode,

Befehle über die Tastatur einzugeben, durch das Verschieben von Symbolen und Menüführung ersetzt. Der Vorteil liegt hier in der sehr leichten Erlernbarkeit der Computerbedienung - die meisten Funktionen erklären sich selbst. Außerdem spart man sich das Blättern in Handbüchern, um diesen oder jenen Befehl zu finden.

Nun ist es natürlich interessant, die Möglichkeiten des GEM in eigenen Programmen zu verwenden. In GEM sind ja alle Funktionen vorhanden, die für die Erstellung von Graphiken und Menüführungen notwendig sind. Doch bevor man drauflos programmiert, sollte man sich ein Bild davon machen, wie GEM arbeitet und wie es aufgebaut ist.

GEM besteht eigentlich aus zwei einzelnen Programmen: dem VDI (Virtual Device Interface) und dem AES (Applikation Environment System). Für beide gelten zwar ähnliche Anwendungsrituale, doch haben sie unterschiedliche Aufgaben.

Das VDI ist zuständig für die Erstellung von Graphiken. Es besteht aus aus den Bestandteilen GDOS (Graphic Device Operating System), das die graphischen Funktionen selbst enthält, und dem GIOS (Graphic Input/Output System). Letzteres ist für die Ein/Ausgabe der Graphik zuständig, es enthält für jedes mögliche Gerät einen spezifischen Gerätetreiber. Dies ermöglicht eine totale Unabhängigkeit vom Gerätetyp, sei es nun der Monitor, der Drucker oder die Diskette, so daß die GDOS-Graphiken auch vom Rechnertyp selbst unabhängig werden. So ist also die Anwendung von Programmen, die auf einem ganz anderen Computer mit GEM geschrieben wurden, möglich!

Das AES ist zuständig für die Benutzerführung, d.h. es verwaltet die Pull-Down-Menüs und die Icons (die Symbole für Diskettenstation, Papierkorb ect.) und die gesamte Fenstertechnik in Verbindung mit der Maus. AES ist also ein mächtiges Betriebssystem.

4.2.1 GEM-Programmierung in BASIC

Diese beiden Teilprogramme des GEM sind für die Programmiersprachen C und Assembler ausgelegt. In dem Entwicklungspaket, das ATARI für Programmentwickler zusammengestellt hat und das leider nicht zum normalen Lieferumfang gehört, sind Bibliotheken der GEM-Parameter enthalten. Der C-Programmierer kann durch einen relativ einfachen Aufruf einer der AES- oder VDI-Routinen eigene Graphiken und Menüs entwerfen und anwenden. In BASIC sieht dies etwas anders aus. Aber die Autoren des BASIC-Interpreters haben die Möglichkeit der GEM-Anwendung auch dieser Sprache glücklicherweise zugänglich gemacht. Der BASIC-Befehlsumfang enthält nämlich die Befehle GEMSYS und VDISYS. VDISYS ist, wie man sich wohl leicht denken kann, ein Aufruf des VDI, wogegen GEMSYS nur das AES anspricht. Mit diesen beiden Befehlen ist es also möglich, das GEM direkt aufzurufen und dies in BASIC-Programme eizubauen. Doch mit einem Aufruf allein ist es nicht getan. Schließlich muß AES bzw. VDI wissen, was es mit welchen Parametern zu tun hat.

Auch daran haben die ATARI-BASIC-Schöpfer gedacht. Wie auch in C gibt es hier Tabellen, in die das Programm diese Parameter übergeben muß. Es handelt sich hier um die Konstanten INTIN, INTOUT, PTSIN, PTSOUT und CONTRL. Diese kann man mit PRINT INTIN,INTOUT,... abfragen. Die erhaltenen Werte sind Speicheradressen, an denen die Parametertabellen beginnen. Um nun z.B. einen Funktionswert in den zweiten Eintrag von INTIN zu legen, muß man zu der INTIN-Adresse 2 addieren, da die Tabellen wortweise aufgebaut sind. Der Tabellenpunkt INTIN(3) hätte also die Adresse INTIN+6.

Diese Tabellen, auch Felder oder Arrays genannt, sind in ihrer Verwendung weitgehend festgelegt. Das INTIN-Array dient zur Übergabe von Eingabeparametern, in INTOUT werden Ausgabewerte abgelegt. PTSIN und PTSOUT werden meist zur Übergabe von Koordinaten verwendet, z.B. bei einer graphischen Ausgabe. Das zur Steuerung der Funktion wichtigste

Feld ist das CONTRL-Array, in dem Werte sowohl einals auch ausgegeben werden. Die Aufteilung ist folgende:

CONTRL(0)	Befehls-Code der Funktion (Funktions-Nummer)
CONTRL(1)	Anzahl der Einträge im PTSIN-Array
CONTRL(2)	Anzahl der Einträge im PTSOUT-Array
CONTRL(3)	Anzahl der Einträge im INTIN-Array
CONTRL(4)	Anzahl der Einträge im INTOUT-Array
CONTRL(5)	Funktions-Kennung (ID) bei Unterfunktionen
CONTRL(6)	Gerätekennung (Handle)
CONTRL(7-n)	Funktionsabhängige Werte

Die Einträge 2, 4 und 6-n sind Ausgabe-Parameter, die die Funktion zurückgibt. Die anderen müssen eigentlich bei jedem Funktionsaufruf eingestellt werden. Der Eintrag in CONTRL(6) ist eine Zahl, die das GEM für das zu bearbeitende Gerät braucht. Da aber der Bildschirm ohnehin geöffnet ist und das aktuelle Gerät darstellt, braucht dieser Kennung nicht übergeben werden. Alle Funktionen werden somit am Bildschirm ausgeführt.

4.2.2 Maus-Abfrage

Wir wollen aus der grauen Theorie einmal ausbrechen und ein kleines Programm schreiben, was die Position der Maus auf dem Bildschirm und gleichzeitig den Zustand der Maustasten feststellt. Dies fällt in die Zuständigkeit des VDI, es findet demnach der VDISYS-Aufruf Verwendung. Nun zum Programm:

```
10 rem *** Get Mouse-Position ***
20 poke contrl,124          : rem Befehlscode
30 poke contrl+2,0          : rem Parameteranzahl
70 vdisys 0                 : rem und Ausführung
80 x = peek(ptsout)         : rem X-Position
```

```
90   y = peek(ptsout+2)      : rem Y-Position
100  taste = peek(intout)    : rem gedrückte Taste
110  print x,y,taste
120  goto 20
```

In Zeile 20 wird zunächst einmal der Befehlscode in CONTRL(0) eingetragen. Dies muß übrigens bei jedem Aufruf von GEM erfolgen. Danach wird in CONTRL(1) die Anzahl der übertragenen Parameter mitgeteilt, was in diesem Beispiel allerdings 0 ist, da keine Eingabewerte für diese Funktion nötig sind. Nun folgt der Aufruf des VDI. Die 0 nach dem VDISYS-Befehl ist willkürlich, irgendeine Zahl muß dort jedoch stehen (ein "Dummy").

Das VDI gibt jetzt seinerseits Werte zurück. Diese Ausgabewerte stehen immer in INTOUT oder PTSOUT, da diese Felder auch nur für Ausgaben vorgesehen sind. In seltenen Fällen erhält man auch im CONTRL-Array Werte geliefert, aber die beiden OUT-Felder sind wichtiger. In dem Beispiel oben wird nun die X- und Y-Position des Mauszeigers in PTSOUT(0) und PTSOUT(1) übergeben. Die Zahl in INTOUT(0) ist 0 bei nicht gedrückter Maustaste. Diese Angaben lassen sich dann im Programm weiterverwerten. Die Zeilen 110 und 120 dienen nur zur Demonstration und können wegfallen.

Da wir gerade beim Mauszeiger sind - ist der Ihnen zu langweilig? Wollen Sie vielleicht ein Männlein über den Bildschirm schieben? Dann müssen Sie dies dem VDI mitteilen. Auch für diese Anwendung habe ich ein Programm vorbereitet, was diesmal auch reichlich Parameter übergibt. Geben Sie also das folgende Programm ein:

4.2.3 Änderung der Maus-Form

```
10   rem *** Set Mouseform - VDISYS-Demo ***
20   poke contrl,111         : rem Befehlscode
30   poke contrl+6,37        : rem Anzahl der Parameter
40   poke intin,3            : rem Aktionspunkt X
50   poke intin+2,0          : rem Aktionspunkt Y
60   poke intin+6,0          : rem Maskenfarbwert
70   poke intin+8,1          : rem Datenfarbwert
80   for i=0 to 15 : read x$ : rem Maske / Cursor
82   x=0 : for j=1 to 16     : rem Umwandlung
84   x= x-(mid$(x$,j,1)<>" ") * 2^(16-j)
86   next j
90   poke intin+10+i*2,x     : rem Maske setzen
92   x=0 : for j=1 to 16     : rem Umwandlung
94   x= x-(mid$(x$,j,1)="*") * 2^(16-j)
96   next j
98   poke intin+42+i*2,x     : rem Cursor setzen
100  next i
110  vdisys 0                : rem und Ausführung
120  end
130  rem +++ Masken-/ Cursordaten +++
140  data "      ...       "
150  data " ..  .***.      "
160  data ".**..*****.     "
170  data " .**..***.      "
180  data "  .** .*.       "
190  data "   .********.   "
200  data "    .*****.**.  "
210  data "    .*****..*.  "
220  data "    .*****.**.  "
230  data "    .**.**....  "
240  data "    .**.**.     "
250  data "    .**.**.     "
260  data "  ...**.**...   "
270  data "  .****.****.   "
280  data "  ...........   "
290  data "                "
```

Dieses Programm bedarf wohl einiger Erklärungen. In Zeile 20 wird wie vorher der Befehlscode der gewünschten VDI-Funktion übergeben. Danach folgt die Angabe der Parameter, die dieser Aufruf übergeben wird. Diese Funktion, deren Name übrigens mit vsc_form festgelegt ist, liest daraufhin 37 Datenworte aus dem INTIN-Array. Die Reihenfclge der Bedeutung jedes Wortes ist ihr bekannt. In INTIN(0) und INTIN(1) liegen die X- und Y-Koordinaten des Aktionspunktes des Mauszeigers, des Punktes also, der z.B. die Pull-Down-Menüs aktiviert. Dieser Punkt wurde in obigem Beispiel auf die Koordinaten 0/3, also die rechte Hand, gelegt.

Die folgenden beiden Werte in INTIN(3) und INTIN(4) geben die Farbwerte des Cursors an. Nun fragt man sich, warum dafür zwei Werte nötig sind. Der Grund dafür ist einfach zu verstehen: angenommen der Cursor ist schwarz und wird auf eine ebenfalls schwarze Fläche geführt; er wäre damit unsichtbar und für eine genaue Positionierung unbrauchbar. Doch nun tritt die Maske in Erscheinung. Diese Maske liegt quasi unter dem Cursor und ist etwas größer als dieser. Ihr Farbwert ist in obigem Beispiel, wie auch im normalen Desktopbetrieb 0, d.h. sie erscheint weiß. Der Effekt ist, daß man auf besagter schwarzer Fläche zwar den Cursor nicht erkennen kann, wohl jedoch seinen Rand. Und dieser Rand ist nichts anderes als die überstehende weiße Maske.

Das Programm überträgt nun ab Zeile 80 die Daten für die Cursor- und die Maskenform. Hier wird wieder das Prinzip der Binär-Dezimal-Umwandlung verwendet, das im Kapitel 2.1 bereits vorgestellt wurde. Für die spezielle Anwendung in diesem Fall wurde die Umwandlungsroutine allerdings modifiziert. So erkennt es nicht mehr nur 1 und 0 als gültige Werte an. Für die Errechnung der Cursordaten, die ja binär das Muster der Form angeben sollen, wird das Sternchen * als 1 anerkannt. Die Maske hingegen wird von den Punkten dargestellt, da sie größer als der Cursor selbst ist. Hier wird ein Leerzeichen als logisch 0 angenommen, alles andere bedeutet für diese Routine zwischen den Zeilen 82 und 90 eine 1. Durch diesen Trick erspart man sich die Aufstellung von zwei unterschiedlichen Tabellen für Cursor und Maske, die ja eigentlich benötigt werden.

Das Umwandlungsprogramm in den Zeilen 80 bis 100 erstellt aus einer kombinierten Tabelle zwei einzelne und überträgt sie direkt in das INTIN-Feld. Beim Aufruf des VDI durch den VDISYS-Befehl stehen somit die Daten für die Maskenform in INTIN(5-20) und die Cursordaten-Tabelle in INTIN(21-36) zur Verfügung. Diese Daten werden dann vom VDI in die interne Tabelle der Zeigerdaten übertragen und dargestellt. Ein Aufruf der Lade- oder Abspeicherungsfunktion der Menüleiste bringt allerdings den Zeiger in die ursprüngliche Form zurück, da während des Diskettenzugriffs die kleine Biene dargestellt wird. Das GEM wählt danach wieder die Pfeilform für die Dateiauswahl.

Wenn Sie nun diese Anwendung in ein Programm einfügen wollen, so ist es natürlich nicht nötig, die anschauliche Binärform der Datentabelle zu übertragen. In diesem Fall ist es empfehlenswert, nach der Erstellung des endgültigen Form des Zeigers die beiden Datentabellen als Dezimalzahlen zu schreiben. Das Programm sähe dann ab Zeile 80 etwa so aus:

```
80  for i=0 to 31          : rem Schleifenbeginn
90  read x                 : rem Wert lesen
100 poke intin+10+i*2,x    : rem und setzen
110 next i
120 vdisys 0               : rem und Ausführung
130 end                    : rem fertig, ggf. RETURN
140 rem --- Maskendaten ---
150 data 384, 2016, 8184, 32766, 65535, 62415, 62415
160 data 62415, 960, 960, 960, 960, 960, 960, 0, 0
170 rem --- Cursordaten ---
180 data 0, 384, 2016, 8184, 29070, 24966, 384, 384
190 data 384, 384, 384, 384, 384, 0, 0, 0
```

Diese Beispieldaten ergeben nicht das Männlein des vorigen Musters, sondern einen senkrechten Pfeil. Der Aktionspunkt müßte hierbei allerdings auf die Koordinaten 8,0 gelegt werden. Wenn Ihnen das Männlein besser gefällt, so können Sie mit Hilfe

des Binär-Dezimal-Umwandlungsprogrammes aus dem Abschnitt 2.1 die entsprechenden Dezimalzahlen ermitteln.

Noch eine abschließende Bemerkung zu dem Beispielprogramm: Ihrer Phantasie sind fast keine Grenzen gesetzt, was die möglichen Formen des Cursors angeht. Beachten Sie jedoch, daß Sie das Format 16x16 beibehalten müssen. Sonst ergeben sich vielleicht noch abenteuerlichere Formen als, Sie eigentlich wollten.

Es gibt allerdings noch eine weitere Möglichkeit, die Mausform zu variieren. Das GEM hat nämlich einige Formen für den Zeiger bereits eingebaut. Dazu gehört auch die kleine Biene, die erscheint, wenn der Rechner arbeitet (z.B. beim Laden). Diese oder andere Form kann durch einen Aufruf des GEM-AES gewählt werden. Das folgende Programm nimmt diese Auswahl vor. Die genaue Erklärung des Programms verschieben wir auf das Kapitel über Fenster- und Menüprogrammierung, in dem das AES im einzelnen vorgestellt wird.

```
10   rem *** Mausform einstellen **
11   defdbl b,d : b=gb
12   cn=peek(b)
14   ii=peek(b+8)
20   d=peek(b+16)          : rem Zeiger erstellen
50   poke cn,78            : rem Befehl
60   poke cn+2,1
70   poke cn+4,1
80   poke cn+6,1
90   poke cn+8,0
100  poke ii,4             : rem Form wählen
110  poke d,257            : rem Form einschalten
120  gemsys 78             : rem und Ausführung
```

Der in Zeile 100 eingesetzte Wert stellt die Form ein. Dabei steht zur Auswahl:

0	Pfeil
1	Cursor
2	Biene
3	Hand mit Zeigefinger
4	flache Hand
5	dünnes Fadenkreuz
6	dickes Fadenkreuz
7	Fadenkreuz als Umriß

Bei der Wahl der Ladefunktion des BASIC oder einer ähnlichen Funktion wird auch hier die Pfeil- bzw. Bienenform wieder eingestellt.

4.2.4 Schriftart-Veränderung

Nun ein weiteres Anwendungsbeispiel, welches diesmal die Schriftart ändert. Das GEM des ATARI ST besitzt die Fähigkeit, Texte auf vielerlei Arten darzustellen. Zu der gewohnten, normalen Schrift kommen noch unterstrichene Zeichen, Fettdruck, hohle Buchstaben und graue Darstellung, die Sie ja von dem BASIC-Editor her kennen. Um diese Variationen anwenden zu können, müssen wir wieder einmal das GEM-VDI bemühen. Ein Programm zur Änderung der Schriftart sieht dann z.B. so aus:

```
10   rem *** Schriftart ändern ***
20   poke contrl  ,106          : rem Befehlscode
30   poke contrl+2,0            : rem Parameteranzahl
40   poke contrl+6,1
50   poke intin   ,1+2          : rem Schriftart
60   vdisys 0                   : rem und Ausführung
70   print "Was für ein Text !" : rem Probetext
80   poke contrl  ,106          : rem nochmal von vorne
90   poke contrl+2,0
100  poke contrl+6,1
110  poke intin   ,0            : rem Normalschrift
120  vdisys 0                   : rem wieder einschalten
```

In diesem Programm wird der Zeichensatz umgeschaltet auf fettgedruckte und graue Schrift. Nach dem Ausdruck einer Probezeile schaltet es wieder auf Normalschrift, um eventuelle Schwierigkeiten mit nachher eingegebenen BASIC-Kommandos zu vermeiden. Die Zeilen 10 bis 40 sind wieder ähnlich den vorhergehenden Beispielen. Die Besonderheit liegt hier in Zeile 50, wo der zu verwendende Zeichensatz definiert wird. Eine Addition ist an dieser Stelle natürlich nicht nötig. Diese Methode erleichtert aber die Schriftarten-Auswahl. Die einzelnen Textmodi sind nämlich bitweise auszuwählen, was eine beliebige Kombination von graphischen Effekten ermöglicht. Die Bedeutung der Bits ist wie folgt:

Bit	Wert	Schriftart
0	1	fett
1	2	grau
2	4	schräg
3	8	unterstrichen
4	16	hohl

Soll also eine graue, schräge Schrift mit Unterstrich dargestellt werden, so ist in Zeile 50 der Wert 2 + 4 + 8, also 14 einzusetzen. Durch die Kombinationsmöglichkeiten können somit 32 Schriftarten gewählt werden. Einige davon, wie z.B. hohle Kursivschrift (16 + 4 = 20), sind allerdings absolut unleserlich, wenn man nicht eine gehörige Portion Phantasie mitbringt...

4.2.5 Graphik-Text

Mit obigen Beispielen haben wir jedoch die Fähigkeiten des VDI noch lange nicht ausgereizt. So können wir mit Hilfe des GEM Textausgaben direkt formatieren lassen und an eine beliebige Stelle des Bildschirms bringen. Dadurch ist man dann in der Lage auch außerhalb des Output-Fensters des BASIC selbst in der Menü-Zeile Texte auszugeben. Die zuständige Funktion des VDI ist hierfür die Nummer 11. Sehen wir uns ein Beispielprogramm an:

```
10   rem *** Graphiktext - VDISYS-Demo ***
15   text$ = "Probetext"
20   poke contrl,11            : rem Befehlscode
30   poke contrl+2,3           : rem Anzahl der Parameter
32   poke contrl+6,len(text$)+2
34   poke contrl+10,10         : rem Funktionskennung
40   poke intin+2,1            : rem Wortdehnung (0=aus)
50   poke ptsin  ,50           : rem X-Koordinate
60   poke ptsin+2,50           : rem Y-Koordinate
70   poke ptsin+4,150          : rem X-Textlänge
80   for i=1 to len(text$)     : rem ASCII-Zeichen
90   poke intin+2+i*2,asc(mid$(text$,i,1)) : rem  setzen
100  next i
110  vdisys 0                  : rem und Ausführung
```

Dieses Programm schreibt den Text aus der Variablen TEXT$ an die Bildschirmkoordinaten 50,50 und dehnt den Text soweit, daß er die vorgegebene Breite von 40 Zeichen erreicht. Die 3, die in Zeile 30 im Control-Array übergeben wird, gibt die Anzahl der in PTSIN zu übernehmenden Parameter an. Die ersten beiden sind die Koordinaten des ersten Buchstabens im Text, wobei 0,0 ganz oben links bedeutet. Die X-Textlänge ist die gesamte Breite, die der Text einnehmen soll. Gedehnt wird der Text durch Einfügen von entsprechend viel Zwischenraum zwischen den Zeichen. Will man den Text ungedehnt an die Stelle X/Y schreiben, so kann man auch den Dehn-Modus ausschalten, indem man in Zeile 40 eine Null übergibt.

In dem Programm ist auch ein wenig Textverarbeitung integriert. Das VDI muß ja wissen, wieviele Zeichen es insgesamt darstellen soll. Diese Anzahl bekommt es in Zeile 32 mitgeteilt, und zwar durch die Angabe der Parameteranzahl, die es aus dem INTIN-Array lesen soll. Die Funktion LEN(TEXT$) gibt die Länge des Textes in dieser Variablen aus, +2 steht für INTIN(0) und INTIN(1). Dann wird der Text Zeichen für Zeichen in das INTIN-Feld eingeschrieben. Um ein Zeichen in den Speicher zu legen, muß man dessen ASCII-Wert einschreiben. ASCII steht für "American Standard of Information Interchange" und ist ein genormter Code, der in vielen Rechnern und Druckern Verwen-

dung findet. Die Funktion ASC("A") gibt in einem BASIC-Programm den ASCII-Wert des A, eine 65, zurück. Jedes Zeichen besitzt einen eigenen Wert, so daß in der Schleife in unserem Beispielprogramm jeder beliebige Text in die Variable TEXT$ eingesetzt werden kann. Das Zeichen, welches gerade umgewandelt werden soll, wird mit der MID$-Funktion aus dem gesamten Text herausgepickt.

Doch wollen wir die Betrachtung der Textverarbeitung auf später (Abschnitt 7.4) verschieben. Bleiben wir also beim VDI, dessen Aufgabenbereich in der Graphik liegt. Die interessantesten VDI-Befehle werden mit Beispielprogrammen im Kapitel 6.1 im einzelnen erklärt. Für eine vollständige Übersicht über die VDI- und AES-Kommandos, deren Anwendung jedoch großteils den C- und Maschinenprogrammierern vorbehalten ist, verweise ich auf das GEM-Buch zum ATARI ST, das ebenfalls bei DATA BECKER erhältlich ist.

Ein interessanter Speicherblock, in dem einige Parameter des BASIC in Verbindung mit GEM stehen, ist durch die SYSTAB-Systemkonstante direkt erreichbar. SYSTAB ist ein Zeiger, der auf diesen Parameterblock weist. Durch PEEK und POKE lassen sich mit dessen Hilfe einige interessante Effekte erzielen. Einige der erreichbaren Parameter sind allerdings nur zum Auslesen geeignet, da deren Änderung leicht zum Absturz führt. Im folgenden sind die Adressen und deren Bedeutung aufgeführt:

SYSTAB	Graphik-Auflösung (1=hoch, 2=mittel, 4=niedrig)
SYSTAB+2	Editor-Bearbeitungs-Schriftart (s.u.)
SYSTAB+4	EDIT-Fenster AES-Zugriffsnummer
SYSTAB+6	LIST-Fenster AES-Zugriffsnummer
SYSTAB+8	OUTPUT-Fenster AES-Zugriffsnummer
SYSTAB+10	COMMAND-Fenster AES-Zugriffsnummer
SYSTAB+12	EDIT-Flag (0=geschlossen, 1=offen)
SYSTAB+14	LIST-Flag (0=geschlossen, 1=offen)
SYSTAB+16	OUTPUT-Flag (0=geschlossen, 1=offen)
SYSTAB+18	COMMAND-Flag (0=geschlossen, 1=offen)
SYSTAB+20	Zeiger auf Graphik-Buffer
SYSTAB+24	GEM-FLAG (0=normal, 1=aus)

Die Graphik-Auflösung ist 1 beim Monochrome-Betrieb und 2 bzw. 4 in Farbe. Ein PEEK(SYSTAB) ermittelt diese Auflösung, was bei graphischen Programmen hin und wieder nötig sein kann, da sich die maximalen X/Y-Koordinaten ändern können.

Mit der Bearbeitungs-Schriftart des Editors ist die normalerweise graue Darstellung einer Zeile gemeint, die gerade bearbeitet wird. Durch ein POKE SYSTAB+2,14 wird beispielsweise diese Darstellung in romanische Schrift geändert. Die Notwendigkeit einer solchen Änderung liegt allerdings im Dunkeln...

Die AES-Zugriffsnummern der einzelnen Fenster des BASIC-Arbeitstisches sind die Nummern, die GEM-AES dem jeweiligen Fenster zugeordnet hat. Mit dieser Nummer kann dann ein AES-Aufruf erfolgen, der dann das gewählte Fenster beeinflußt (z.B. verschieben, vergrößern oder verkleinern).

Die Flags der Fensterzustände enthalten Zustandsinformationen über die jeweiligen Fenster. Das GEM erfährt hier, ob das Fenster überhaupt existiert. Die Manipulation dieser Flags ist nicht ungefährlich, da ein LIST-Befehl zum Absturz führen kann, wenn das LIST-Flag gelöscht wurde. Diese Flags sollten daher nicht verändert werden.

Die Speicherzellen SYSTAB+20 bis SYSTAB+23 enthalten in einem 32-Bit-Langwort die Speicheradresse des Graphik-Buffers. Dieser Buffer gewährleistet die Erhaltung des Inhaltes des OUTPUT-Fenster bei Verändern seiner Größe. Durch Anwahl des Menüpunktes Buffered Graphics läßt sich durch Anklicken dieses Punktes die Extra-Speicherung dieser Graphiken ausschalten. Dadurch gewinnt man für den BASIC-Arbeitsspeicher immerhin 32 KByte. Wenn die Option eingeschaltet ist, so belegt sie nämlich genausoviel wie der ganze Bildschirm. Die Adresse dieses Speicherbereiches läßt sich durch Auslesen des Langwortes ab SYSTAB+20 feststellen.

Und nun noch zum GEM-Flag. Wird dieses mit POKE SYSTAB+24,1 auf 1 gesetzt, so ist das GEM abgeschaltet. Das merkt man nicht sofort, es verschwinden also keine Fenster, aber eine Veränderung eines Fensters ist nicht mehr möglich.

Ebenso wird keine Eingabe von der Tastatur mehr akzeptiert. Das GEM-Flag darf also nur innerhalb eines Programmes verwendet werden, welches es bei Ein- und Ausgaben bzw. bei Programmbeendigungen wieder auf 0 setzt. Der Vorteil dessen, das Flag überhaupt zu setzen, liegt in der Zeit, die das GEM normalerweise vom Prozessor beansprucht, um Fenster- und Menü-Überwachung zu realisieren. Diese Überwachung wird bei Ausschalten des GEM eingestellt, und der Prozessor kann der Bearbeitung des BASIC-Programmes mehr Zeit widmen. Das Programm läuft dann schneller, was bei langwierigen Berechnungen manchmal ganz angenehm sein kann. Diskettenzugriff ist, da er vom BIOS bearbeitet wird, weiterhin erlaubt. Aber nicht vergessen: GEM wieder einschalten!

4.3 Interpreter/Compiler

Wußten Sie, daß ein Computer nur eine Sprache, nämlich die Maschinensprache, spricht? Aber der ATARI ST kann doch auch LOGO und BASIC, werden Sie jetzt sagen. Das stimmt auch, aber doch nur, wenn diese Programmiersprache in den Rechner geladen wurde. Dieses Programm, z.B. BAS.PRG, ist ein Übersetzer, der die Eingaben in der Sprache BASIC in den Maschinencode umarbeitet. Diese Übersetzer machen den ST dann erst zu einem BASIC-, LOGO- oder C-Rechner.

Man unterscheidet zwei verschiedene Arten solcher "Dolmetscher": Interpreter und Compiler. Bei dem Beispiel BASIC handelt es sich um einen Interpreter. Hier werden BASIC-Programme Zeile für Zeile übersetzt. Allerdings wird nur die Programmzeile interpretiert, die gerade an der Reihe ist. Dadurch ist ein Nachteil von Interpretern schon klar: sie sind langsam. Wenn wir ein Beispielprogramm in BASIC betrachten, wird das klar. Nehmen wir dafür eine einfache Schleife:

```
10   FOR I = 1 TO 100
20   A = A * 3
30   NEXT I
```

Diese Schleife läßt 100 mal die Variable A verdreifachen. Der Interpreter muß dafür nicht nur 100 mal die Laufvariable I um eins erhöhen und entscheiden, ob die Schleife beendet ist. Er muß auch jedesmal, wenn er die Zeile 20 liest, diese interpretieren. So ist das ja nur ein Platzhalter, dessen wahre Speicheradresse erst ermittelt werden muß. Außerdem ist die 3 ein Wert, den er jedesmal für die Verrechnung mit A in ein Fließkommaformat umwandelt. All dies geschieht zwar schnell, aber oft. Somit wird für die eigentliche Arbeit der Verdreifachung von A nur ein kleiner Teil der wirklich gebrauchten Zeit verwandt.

Im Gegensatz zu dieser Methode der Programmübersetzung stehen die Compiler. Die Programmiersprache C verwendet ein solches Prinzip, wodurch ein fertig compiliertes C-Programm sehr schnell wird. Hierbei wird ein Programm, welches anfangs als Quelltext vorliegt, nur einmal übersetzt. Nach diesem Vorgang, dem Compilieren, liegt ein neues Programm vor, welches zwar dieselbe Funktion hat wie das Quellprogramm, aber vollständig aus Maschinencode besteht. Dadurch wird ein compiliertes Programm natürlich schneller, jedoch muß nach jeder Änderung der Compilierungsvorgang wiederholt werden. Ein Programmierer, der das Arbeiten mit BASIC gewohnt ist, wird daher beim Umstieg auf eine Compilersprache wie C eine wesentlich höhere Arbeitsdisziplin brauchen. Das Trial and Error Prinzip einer Programmerstellung in BASIC, also erst einmal drauflosschreiben und hin und wieder RUN tippen, ist hier sehr zeitaufwendig. Das Ausprobieren eines C-Programmes erfordert nicht nur Zeit und somit Geduld, sondern auch gute Nerven - kann ein Maschinenprogramm doch bei einem Fehler den Rechner zum Absturz bringen.

Allerdings hat auch ein Compiler seine Nachteile. Meistens wird zu dem selbstgeschriebenen Programm eine ganze Bibliothek von Hilfsroutinen zugefügt, die beim Compilieren vom sogenannten Linker angehängt werden. Diese Hilfsroutinen, bei C z.B. Ein-

/Ausgabefunktionen, sind aber in den meisten Fällen mit viel mehr Fähigkeiten ausgestattet als, man benötigt. Dadurch kommt es dazu, daß fertig compilierte Programme länger sind als eigentlich nötig wäre. Und lang sind diese Programme ohnehin. Man braucht sich nur den Umfang des BASIC-Interpreters anzusehen, der selbst in einer C-ähnlichen Sprache geschrieben wurde. Würde man dieses Programm direkt in Maschinensprache schreiben, hätte es nur noch einen Bruchteil des Umfanges. Nur ist eine Programmerstellung in Maschinensprache wesentlich komplizierter und damit zeitaufwendiger.

Damit sind wir bei der Frage, welche Programmiersprache für welchen Zweck am besten ist. Pauschal läßt sich diese Frage nicht beantworten, da es dabei zu viele Kriterien zu beachten gibt. Zum einen wäre da die nötige Geschwindigkeit, mit der das Programm arbeiten soll. Bei zeitkritischen Aufgaben, wie zum Beispiel der Meßwerterfassung sich ständig ändernder Größen in Echtzeit, ist eine Interpreterorientierte Sprache wie BASIC sicher nicht besonders geeignet.

Andererseits spielt es eine Rolle, wie oft das Programm voraussichtlich geändert werden muß. Will man den Computer irgendwelche Berechnungen anstellen lassen, deren Ergebnis man nur einmal braucht, so ist dies in einer einfachen Sprache schneller programmiert. Ein C-Programm zu schreiben, zu Compilieren und zu Linken ist für eine solche Anwendung wohl meistens viel zu viel Aufwand.

Zusammengefaßt gesagt, ist es sicher ideal, Programmiersprachen beider Arten zu beherrschen. Beim ATARI ST bieten sich hier der BASIC-Interpreter und der C-Compiler an. Welche dieser beiden Sprachen Sie schließlich für dieses oder jenes Problem zur Lösung verwenden, können Sie dann für jeden Fall nach obigen Überlegungen entscheiden.

Ausgabefunktionen, sind aber in den meisten Fällen mit viel mehr Fähigkeiten ausgestattet als man benötigt. Dadurch kommt es dazu, daß fertig compilierte Programme länger sind als es eigentlich nötig wäre. Und lang sind diese Programme schon. Man braucht sich nur den Umfang des BASIC-Interpreters anzusehen, der selbst in einer C-ähnlichen Sprache geschrieben wurde. Würde man dieses Programm direkt in Maschinensprache schreiben, hätte es nur noch einen Bruchteil des Umfanges. Nur ist eine Programmerstellung in Maschinensprache wesentlich komplizierter und damit zeitaufwendiger.

Nun sind wir bei der Frage, welche Programmiersprache für welchen Zweck am besten ist. Pauschal läßt sich diese Frage nicht beantworten, da es dabei zu viele Kriterien zu beachten gibt. Zum einen wäre da die nötige Geschwindigkeit, mit der das Programm arbeiten soll. Bei zeitkritischen Aufgaben, wie zum Beispiel der Meßwerterfassung sich ständig ändernder Größen in Echtzeit, ist eine Interpreterorientierte Sprache wie BASIC sicher nicht besonders geeignet.

Andererseits spielt es eine Rolle, wie oft das Programm voraussichtlich gestartet werden muß. Will man den Computer irgendwelche Berechnungen anstellen lassen, deren Ergebnis man nur einmal braucht, so ist dies in einer einfachen Sprache schneller programmiert. Ein C-Programm zu schreiben, zu Compilieren und zu Linken, ist für eine solche Anwendung wohl meistens viel zu viel Aufwand.

Zusammengefaßt gesagt, ist es sicher ideal, Programmiersprachen beider Arten zu beherrschen. Beim ATARI ST bieten sich hier der BASIC-Interpreter und der C-Compiler an. Welche dieser beiden Sprachen Sie schließlich für dieses oder jenes Problem zur Lösung verwenden, können Sie dann für jeden Fall nach obigen Überlegungen entscheiden.

5. Arbeitsvorbereitung

Bevor man mit der Arbeit am ATARI ST anfängt, sollte man erst einmal seinen Arbeitsplatz in den Zustand bringen, mit dem man fortan arbeiten will. Dazu gehört das richtige Aufräumen des Schreibtisches auf dem Bildschirm, aber auch die Definition der verschiedenen Schnittstellen-Parameter. Hat man einmal all dies erledigt, so kann man mit dem Menüpunkt *Arbeit sichern* den aktuellen Zustand seines Arbeitsplatzes auf Diskette ablegen. Nach jedem Reset wird dann dieser Zustand wieder eingestellt, indem das Betriebssystem die Datei DESKTOP.INF lädt. In dieser Datei stehen alle Einstellungen, die beim Sichern der Arbeit aktuell waren. Diese Einstellungen sind allerdings nur im vorgegebenen Rahmen möglich. Doch diesen Rahmen kann man sprengen, da der ST mehr kann als z.B. bei der Konfiguration der Schnittstelle aus dem Desktop-Menü geboten wird. Sehen wir uns das einmal etwas genauer an:

5.1 Einstellung der Schnittstelle

Will man die Übertragungsrate der seriellen Schnittstelle einstellen, so bekommt man die Auswahl von Werten zwischen 300 und 9600 Baud. Dies sind die gebräuchlichen Baudraten bei serieller Kommunikation. Doch gibt es hin und wieder den Fall, daß man eine schnellere bzw. langsamere Übertragung wünscht. Der im ST eingebaute Chip, welcher für die Seriellschnittstelle zuständig ist, ist dazu bestens in der Lage. Er beherrscht Geschwindigkeiten von 50 bis zu 62500 Baud, was nun wirklich für alle Anwendungen der RS232-Schnittstelle ausreicht.

Nun gibt es für diesen Bereich allerdings schon eine Grenze: das Betriebssystem unterstützt nur die Raten 50 bis 19200 Baud. Wollen Sie einen der Werte einstellen, der außerhalb der üblichen Geschwindigkeiten liegt, so müssen Sie das dem Betriebssystem selbst mitteilen. Dies geht jedoch nur über die Maschinensprache. Also müssen wir ein kleines Maschinenprogramm schreiben.

Der einfachste Weg hierfür wäre die Verwendung eines Assemblers, der im Entwicklungspaket von ATARI enthalten ist. Doch nicht jeder besitzt dieses Programm, also machen wir den Umweg über das BASIC.

Das folgende Programm erstellt ein Programm namens SETBAUD.PRG und legt dies auf der Diskette ab. Um es aufzurufen, braucht es nur auf gewohnte Art vom Desktop aus aufgerufen zu werden. Danach ist die Seriell-Schnittstelle nach Ihren Wünschen konfiguriert. Ein Aufruf des Menüpunktes RS-232-Einstellung setzt die Schnittstelle sofort wieder in den dort eingestellten Zustand zurück. Nun zum Programm:

```
10   rem *** Baudrate konfigurieren ***
20   for i=1 to 72  step 2          : rem 36 Worte
30   read a                         : rem lesen
40   poke i+1999,a                  : rem in den Speicher
50   next i                         : rem ablegen
60   bsave "SETBAUD.PRG",2000,72    : rem und abspeichern
70   end
80   rem  --- Programmdaten ---
100  data 24602,0,44,0,0,0,0,0,0,0,0,0,0,0
110  data 16188,-1,16188,-1,16188,-1
120  data 16188,-1,16188,-1,16188,15
130  data 16188,15,20046,-8196,0,14
140  data 16999,16188,76,20033
```

Das so erstellte Programm setzt die Übertragungs-Rate auf 50 Baud. Ist eine andere Geschwindigkeit gewünscht, so muß man die entsprechende Codezahl statt der 15 am Ende der Zeile 120 eintragen. Diese Zahl können Sie der folgenden Tabelle entnehmen:

Kennzahl	Baudrate
0	19200
1	9600
2	4800
3	3600
4	2400
5	2000
6	1800
7	1200
8	600
9	300
10	200
11	150
12	134
13	110
14	75
15	50

Alle anderen Einstellungen der Schnittstelle, wie das Übertragungs-Protokoll oder die Parität, bleiben im eingestellten Zustand erhalten.

Wollen Sie zusätzlich das Übertragungsprotokoll ändern, so können Sie das mittels der zweiten -1 in Zeile 120 bewerkstelligen. Hier die einzelnen Bedeutungen:

Wert	Bedeutung
-1	eingestellter Wert bleibt erhalten
0	kein Handshake
1	XON / XOFF
2	RTS / CTS
3	beides. Ist aber nicht sinnvoll.

Noch ein Hinweis zur Anwendung dieser Einstellung: das Betriebssystem initialisiert bei einigen Funktionen die Schnittstelle neu. Es werden dann wieder die Parameter der Menü-Einstellung übernommen und damit die alte Baud-Rate eingestellt. Benennt man das zuständige Programm für diese Einstellungen, DESK1.ACC, um (z.B. DESK1.XXX) und startet das System neu, dann ist diese Einstellmöglichkeit weg. Die Baud-

rate, die mit unserem Programm eingestellt wird, bleibt dann erhalten.

Es gibt aber eine weitere Möglichkeit der Einstellung. Wenn Sie in einem BASIC-Programm die Baud-Rate variieren wollen, so können Sie das mit Hilfe des CALL-Befehls und eines Maschinenprogrammes. CALL heißt soviel wie rufen, und das tut dieser Befehl auch. Er ruft ein Maschinenprogramm auf, welches allerdings für die Anwendung aus BASIC-Programmen vorgesehen sein muß. In Kapitel 6.4 erfahren Sie anhand von Beispielen, wie diese Kombination zweier Programmiersprachen funktioniert.

6. Programmiersprachen

Für einen Computer vom Kaliber des ATARI ST muß eine Programmiersprache einen enormen Umfang haben, um all die Besonderheiten des Rechners auch bedienen zu können. Aus diesem Grunde war für ATARI in Punkto Sprachen das beste gerade gut genug. Die momentan verfügbaren Sprachen sind alle sehr komfortabel und decken den ersten Bedarf von ST-Besitzern völlig ab. Einige dieser Übersetzer wollen wir nun unter die Lupe nehmen und uns ansehen, welche Vor- und Nachteile sie im einzelnen für unsere Zwecke haben. Beginnen wir mit den Interpretern LOGO und BASIC, die für die einfacheren Anforderungen sehr gut geeignet und außerdem recht leicht zu handhaben sind.

6.1 Dr. LOGO

Der oft wie in der Überschrift geschriebene Name dieser Programmiersprache bedeutet nicht etwa, daß es sich dabei um eine Sprache für Ärzte handelt. DR LOGO, wie es richtig heißt, ist die Abkürzung von Digital Research LOGO und wird in der beiliegenden Beschreibung auch als ATARI-LOGO bezeichnet. Dieses Programm, welches in ähnlicher Form auch auf anderen Rechnern läuft, ist eigentlich zu Unrecht als eine Kindersprache bekannt. In Wirklichkeit hat man mit DR LOGO eine sehr komfortable und fähige Sprache vor sich.

LOGO hat ein wenig Ähnlichkeit mit FORTH, die man gerne als die unbegrenzte Programmiersprache bezeichnet. Der Grund dafür ist die Möglichkeit, beliebige eigene Befehle selbst zu definieren. Und das geht auch in LOGO. Nehmen wir hierfür ein Beispiel.

Will man aus einem Programm heraus Daten an den Drucker ausgeben, so ist das nicht ohne weiteres möglich. LOGO verfügt lediglich über den Befehl COPYON bzw. COPYOFF, der alle Ausgaben auf den Drucker leitet. Möchte man das Kommando LPRINT haben, welches von BASIC her bekannt ist, so definiert

man es sich ganz einfach selbst. Das sieht dann folgendermaßen aus:

```
TO LPRINT :ZEILE
  COPYON  PR :ZEILE COPYOFF
END
```

Ein solches Programm, das durch seinen Namen aufgerufen wird, nennt man Prozedur. Die Prozedur LPRINT[a] ist nun genauso verfügbar wie die anderen, resistenten Befehle des LOGO auch. Einen Text gibt man dann einfach mit LPRINT[Text] aus.

Dieses Beispiel deutet auch schon auf das Prinzip der LOGO-Programmierung hin. Für die einfachen Aufgaben eines Programmes werden Prozeduren geschrieben. Diese werden dann ihrerseits von übergeordneten Prozeduren verwendet und so weiter, bis man das ganze Programm schließlich auf einen einzigen Befehl reduziert hat. Diese hierarchische Struktur hat den Vorteil, daß man sehr übersichtlich programmieren kann, erfordert aber auch eine gewisse Disziplin beim Programmaufbau. Ein unsystematisch aufgebautes Programm ist später nur noch schwerlich lesbar.

Ein weiterer Punkt der LOGO-Philosophie ist die Listenorientierung der Sprache. So werden die meisten Daten in Form von Listen verwaltet, deren Verarbeitung sehr umfangreich unterstützt wird. Listen können aus beliebig vielen Zahlen oder auch Texten bestehen und durch einen einzigen Befehl sortiert, gemischt, untersucht oder verändert werden. Somit eignet sich diese Sprache für Textverarbeitung oder Tabellenkalkulation. Aber noch eine andere Spezialität hat LOGO: die Graphik. Turtle-Graphik nennt man sie auch, da der zeichnende Cursor in manchen LOGO-Versionen als Schildkröte (= Turtle) dargestellt wird. Beim ATARI ST ist es ein Dreieck, was sich durch Kommandos wie LEFT, RIGHT, FORWARD oder BACK in Bewegung setzen läßt. Wenn es gewünscht wird, dann hinterläßt es einen Strich, wodurch sich leicht Graphiken herstellen lassen. Dafür ein Beispiel:

```
TO BROSCHE
 FD 100 RT 111 FD 40
  TURN
  BROSCHE
END

TO TURN
  FD 66 RT 66 FD 66
END
```

Dieses Beispiel zeichnet bei dem Befehl BROSCHE ein Ornament auf den Graphik-Bildschirm. Da die Prozedur BROSCHE sich selbst immer wieder aufruft, läuft es solange, bis es gestoppt wird oder der Strom ausfällt.

Doch eine erschöpfende Beschreibung der Programmiersprache DR LOGO würde wohl den Rahmen dieses Buches sprengen. Für Interessierte gibt es eine Anzahl guter Bücher, die in diese Sprache einführen. Wir wollen uns aber lieber ansehen, wie LOGO das Betriebssystem überlisten kann.

Auch hier gibt es die PEEK- und POKE-Funktionen, nur daß sie hier .EXAMINE und .DEPOSIT lauten. .EXAMINE 1102 gibt z.B. die Adresse des Bildschirmspeichers an, während .DEPOSIT 1262,0 den Ausdruck des Bildschirms startet.

6.2 ATARI-BASIC

BASIC ist wohl unter den Hobby-Computeristen die beliebteste Sprache überhaupt. Das liegt daran, daß diese Sprache leicht zu erlernen und nicht besonders kritisch ist, was den Programmierstil angeht. Ein BASIC-Programm ist schnell geschrieben, Fehler findet man durch das Trial-and-Error-Prinzip schnell, da man einen beim Ablauf auftretenden Fehler direkt korrigieren kann. Dies ist bei Sprachen wie C oder Maschinensprache nicht so leicht möglich.

Der ATARI-BASIC-Interpreter speichert die eingegebenen Programmzeilen als Text ab, der nach dem Start des Programms Zeile für Zeile interpretiert wird. Viele Interpreter arbeiten etwas anders. Bei ihnen wird ein BASIC-Befehl als Codezahl, Tokens genannt, abgespeichert. Diese Methode vereinfacht die Befehlsinterpretation und erhöht dadurch die Geschwindigkeit des Programms. Doch das ATARI-Basic ist auch ohne die Vorübersetzung in Tokens so schnell, daß so mancher erfahrener BASIC-Programmierer staunt.

Ein BASIC-Programm kann von der Diskette direkt in einen Texteditor geladen und dort verarbeitet werden. Dies ist jedoch nicht nötig, da der BASIC-Editor einen hohen Komfort bietet. Es stört lediglich, daß die TAB-Funktion nicht erlaubt ist. Das ATARI-BASIC zeichnet sich außerdem noch durch die angenehme Syntax-Test-Funktion aus, die eine eingegebene Programmzeile sofort auf ihre Richtigkeit prüft. Der Nachteil ist, daß in dem BASIC-Editor keine normalen Texte editiert werden können, ohne daß man nach Eingabe jeder Zeile eine Fehlermeldung "Something is wrong" bekommt. Daher sollte man für die einfache Textverarbeitung lieber auf den LOGO-Editor zurückgreifen.

Im Gegensatz zu den anderen Sprachen, die hier besprochen werden, ist BASIC zeilenorientiert. Das bedeutet, daß jede Programmzeile eine Nummer hat, auf die der GOTO- bzw. GOSUB-Befehl zugreift. Will man aber einen Programmteil irgendwo in das bestehende Programm einfügen, so muß dafür genügend Platz sein. Aus diesem Grund ist es empfehlenswert, bei der Programmerstellung einen Abstand von 10 Zeilen zu wählen. Der Befehl AUTO, der automatisch für die Programmierung Zeilennummern vorgibt, nimmt ebenfalls diese Schrittweite an, wenn sie nicht anders angegeben ist.

Hat man dennoch mehr als neun Zeilen einzufügen, so kann ein Trick helfen. Man läßt mit dem RENUM-Befehl die Zeilennummerierung neu vornehmen, wobei man eine genügend hohe Schrittweite vorgibt. Danach kann man seinen Programmteil eingeben und nachher ggf. mit normalem Zeilenabstand renummerieren.

BASIC ist keine problemorientierte Sprache. Während COBOL für wirtschaftliche Aufgaben und FORTRAN für naturwissenschaftliche Probleme vorgesehen ist, ist BASIC für alle Aufgaben einzusetzen. Es bietet zwar keine speziellen Befehle, wie Sortieren oder Mittelwertbildung, kann diese jedoch durch Unterprogramme nachbilden. Somit ist nahezu jede Aufgabe mit einem BASIC-Programm zu erledigen. Problematisch ist jedoch die Verarbeitung von sehr schnell ablaufenden Prozessen. Für solche zeitkritischen Aufgaben ist die Maschinensprache wesentlich geeigneter. Man kann allerdings diese beiden Sprachen miteinander kombinieren, was wir in Kapitel 6.4.1 betrachten wollen.

Dieses Buch ist ebenfalls für BASIC-Programmierer geschrieben, die die phantastischen Möglichkeiten des ATARI ST mit dieser Sprache vollständig ausnutzen wollen. Das ist zwar nicht in vollem Maße möglich, doch bietet BASIC mit seinen Systembefehlen wie PEEK, POKE, CALL oder USR die Möglichkeit das Betriebssystem oder Maschinenspracheprogramme mit in ein Programm einzubeziehen. Aus diesem Grund heißt das Buch auch nicht "EXAMINEs und DEPOSITs", wie PEEK und POKE in LOGO genannt werden, da nur BASIC eine einfache Programmierung mit der Möglichkeit der Einbeziehung des gesamten Computersystems bietet.

6.3 C

Die Programmiersprache C ist eine Hochsprache, deren merkwürdiger Name daher kommt, daß die Entwickler dieser Sprache ihre Versionen einfach mit Buchstaben benannten. Zuerst kam also A, dann B und schließlich kam C, heute eine weit verbreitete Sprache, die in sehr vielen Computersystemen Verwendung findet.

C ist eine sehr maschinennahe Sprache. Dies äußert sich schon darin, daß Ein-/Ausgaberoutinen nicht zum normalen Sprachumfang gehören. Der Grund dafür ist, daß gerade diese Routinen systemspezifisch, also abhängig vom verwendeten Computer selbst sind. Es können natürlich trotzdem Eingaben und Aus-

gaben gemacht werden. Dafür gehört zum Lieferumfang jedes C-Compilers eine Library, eine Bibliothek mit all diesen spezifischen Funktionen. Durch das Verbinden des eigentlichen Programms mit Routinen aus der Library können dann alle möglichen Funktionen in das Programm eingefügt werden. Diese Verbindung stellt der Linker her, ein Hilfsprogramm, das ebenfalls bei jedem C-Compilerpaket vorhanden ist.

Der Vorteil dieses eher spartanischen Befehlsvorrates, den diese Sprache bietet, ist der, daß C-Programme nahezu ohne Änderung von einem Computer auf einen anderen übertragbar ist. Dadurch kann ein Programm, oder gar ein Betriebssystem, welches einmal geschrieben wurde, auch leicht auf einen neuen Computertyp übertragen werden.

Der ATARI ST zeigt deutlich wie günstig das ist. Schließlich wurde das GEM, welches in dem Betriebssystem des ST enthalten ist, ebenfalls in C geschrieben. Die gleiche Version des GEM gibt es daher auch für einige andere Computer, wie zum Beispiel den IBM-PC.

Auch das mächtige Betriebssystem UNIX, welches vor allem in Großrechenanlagen Verwendung findet, ist in C geschrieben. Man sieht also, daß diese Sprache sehr interessant ist!

Für eine detaillierte Beschreibung von C fehlt uns hier der Platz, da die C-Philosophie recht komplex ist. Lassen Sie mich daher nur einen kleinen Überblick dieser Programmiersprache vornehmen.

C stammt aus der Familie der ALGOL-Sprache. Aus dieser Sprachfamilie sind ebenfalls FORTRAN und PASCAL entstanden, Sprachen, die sich durch ihre hohe Überschaubarkeit und durch strukturierten Aufbau auszeichnen. Man kann hier sogenannte Prozeduren definieren, deren Aufruf wie ein Befehl vorgenommen wird. Die Prozedur PRINTF ist eine davon. Sie gehört zu den Standard-Input/Output-Routinen, die in der oben erwähnten Library zur Verfügung gestellt wird. Diese Prozedur gibt, ähnlich dem PRINT-Befehl in BASIC, einen Text oder Zahlenwerte aus. Dieser Funktion werden die Texte bzw. Werte,

die sie ausgeben soll, übergeben, sowie die Anweisung, in welchem Format diese Ausgabe erfolgen soll. Die Übergabe von Parametern sieht in C folgendermaßen aus:

```
PRINTF("%5.2f %6.1f\n",A,B)
```

Die Argumente, die mit einem %-Zeichen beginnen, bestimmen das Format der Ausgabe. %5.2f bedeutet hier, daß der Wert der Variablen A als Gleitkommazahl mit mindestens 5 Zeichen Breite und 2 Nachkommastellen dargestellt werden soll. Das f steht hierbei für Fließkommadarstellung (floating point).

So kompliziert dieser Ausdruck auch aussieht, so komfortabel ist er auch. Aus diesem Grund wird er auch häufig verwendet.

Doch bleiben wir erst einmal bei dem normalen Sprachumfang von C. Dieser enthält nur einfache Kontrollstrukturen, wie Entscheidungen, Schleifen, Zusammenfassungen und Unterprogrammen. Variablen müssen am Programmanfang grundsätzlich definiert werden, um den Datentyp festzulegen. Eine leichte und unkomplizierte Programmierung wie in BASIC wird hier nicht zugelassen, was aber die Fehlersuche in einem Programm erleichtert.

Die Struktur eines C-Programmes ist nun folgende: Zuerst werden die Variablen definiert, die das Programm benötigt. Dann werden einzelne Prozeduren geschrieben, die alle voneinander getrennt sind. Um nun den Ablauf eines C-Programmes nachvollziehen zu können, sucht man zuerst die Routine heraus, die den Namen MAIN() trägt. Hierbei handelt es sich um das eigentliche Programm, welches üblicherweise eine Reihe von Prozeduraufrufen enthält. Da die Benennung dieser Prozeduren meist auf ihre Funktion schließen läßt, kann so die grobe Struktur des Gesamtprogrammes leicht überschaut werden.

Kleine Programme bestehen meist nur aus der MAIN-Routine. Sehen wir uns ein solches kleines C-Programm einmal an.

```
/* Umwandlung von Radius in Umfang */

main()
 {     int von, bis, step;
       float radius, umfang;

       von = 1;          /* Anfangsradius */
       bis = 10;         /* Endradius */
       step = 1;         /* Schrittweite */

       radius =von;
       while (radius <= bis)
       {
                umfang = 2 * radius * 3.141592;
                printf("%2.0f %7.3f\n",radius,umfang);
                radius = radius + step;
       }
 }
```

Die erste Zeile stellt einen Kommentar dar. Ein Kommentar wird in die Zeichen /* und */ eingeschlossen und vom Compiler ignoriert. Danach beginnt unser Programm mit der MAIN-Routine. Die leere Klammer deutet darauf hin, daß diese Funktion keine Parameter übernehmen kann.

Nun folgen die Definitionen der Variablen. Hierbei stehen folgende Datentypen zur Verfügung:

int	Integerzahlen
float	Fließkommazahlen
char	ein einzelnes Zeichen
short	ein kleiner, ganzzahliger Wert
long	ein großer, ganzzahliger Wert
double	ein Gleitkommawert mit doppelter Genauigkeit

Unser Programm weist dann den Variablen Anfangswerte zu. Danach beginnen wir in einer Schleife die 10 Umfangswerte zu berechnen. Diese Schleife wird durch WHILE(Bedingung) definiert und läuft solange, wie diese Bedingung erfüllt wird. Ein

WEND, wie es das BASIC für die Beendigung einer WHILE-Schleife benötigt, gibt es hier nicht. Es wird hier der Programmteil, der in der folgenden Klammer steht, wiederholt.

Dieser Programmteil errechnet nun aus dem Radius den Umfang eines Kreises und gibt beide Werte formatiert aus. Danach wird der Radius um 1 erhöht und die Schleife weitergeführt, wenn er den Endwert 10 nicht überschritten hat. Andernfalls wird das Programm beendet.

Wie man sieht, ist ein C-Programm nicht so fremd, wenn man mal irgendeine Programmiersprache gelernt hat. In C werden alle Anweisungen mit einem Semikolon abgeschlossen wie in PASCAL auch. Die Befehle IF, ELSE, WHILE oder FOR kommen einem direkt bekannt vor. Die teilweise umständliche Programmierung wird aber durch übersichtliche Strukturierung relativ einfach lesbar, und der Arbeitsaufwand wird durch eine sehr hohe Arbeitsgeschwindigkeit des fertigen Programmes belohnt.

Für den ATARI ST gibt es einen weiteren Grund zur Verwendung von C. Die Programmierung des GEM ist nämlich für C-Programme ausgelegt, so daß in dieser Sprache die Entwicklung von graphisch unterstützten Programmen sehr einfach wird. In dem Entwicklungspaket von ATARI ist eine riesige Bibliothek von GEM-Befehlen und Systemparametern enthalten, die dem C-Programmierer die vollen Möglichkeiten des GEMDOS zugänglich macht.

6.4 Maschinensprache des MC 68000

In der Reihe der Programmiersprachen nimmt die Maschinensprache eine absolute Sonderstellung ein. Diese Sprache ist zugleich die primitivste und mächtigste Art, einen Computer zu programmieren. Dieser scheinbare Widerspruch ist leicht zu erklären:

In jedem Computer befindet sich eine zentrale Recheneinheit, die CPU (Central Prozessor Unit). Dieses Gehirn des Rechners ist derjenige Baustein, der eigentlich die gesamte Arbeit erledigt, die dem Rechner aufgetragen wird. Programmiert man in einer Hochsprache wie BASIC oder LOGO, so wird das, was man an Befehlen eingegeben hat, in eine Sprache übersetzt, die der Prozessor versteht. Diese Sprache ist die Maschinensprache. Der Vorteil, den sie gegenüber allen anderen Sprachen besitzt, ist der, daß sie alle Funktionen des gesamten Computers steuern kann. Genau da liegen ja die Einschränkungen, denen z.B. BASIC unterliegt. Dort ist man gezwungen für einige Operationen den Umweg über die Maschinenebene zu nehmen.

Auf eine dieser Einschränkungen sind wir bereits in dem Abschnitt gestoßen, wo es darum ging, die serielle Schnittstelle nach unseren Wünschen zu konfigurieren. Dabei war es selbst mit dem Allround-Befehl POKE nicht getan, so daß wir ein Maschinenprogramm schreiben mußten. Aber das ist auch nur die Spitze des Eisberges.

Ein weiterer Vorteil der direkten Programmierung des Prozessors liegt in der Geschwindigkeit, die damit zu erreichen ist. Man braucht sich nur einmal das Tempo anzusehen, mit dem bei einem Programmaufruf der Bildschirm gelöscht wird. Ein Programm in BASIC sieht hierfür so aus:

```
10   defdbl a,b : b = 1102
20   a = peek(b)
30   for i = a to a + 32768
40   poke i,0
50   next i
```

Wer ein gewisses Augenmaß für die Geschwindigkeit eines BASIC-Programmes besitzt, wird sofort sehen, daß selbst ein schnelles BASIC, wie das des ST, für dieses Programm einige Sekunden benötigen würde. Man könnte hier zusehen, wie der Bildschirm Zeile für Zeile gelöscht wird. Ein Löschvorgang, den

das Betriebssystem vornimmt, geht dagegen schneller, als man es verfolgen kann.

Sicher wird mancher jetzt denken, daß diese Vorgänge vom Betriebssystem unterstützt werden, so daß man sich nicht um solche Zeitprobleme kümmern muß. Aber wie ist es denn bei einer Anwendung wie z.B. einer Textverarbeitung, wo ein Sortiervorgang nicht unbedingt eine Kaffeepause erzwingen darf?

Sie sehen, an der Maschinensprache kommt man nicht vorbei, wenn man das System vollkommen ausschöpfen will. Und gerade der im ST eingebaute Prozessor MC 68000 bietet in Verbindung mit einem so mächtigen Betriebssystem wie GEMDOS enorme Möglichkeiten.

Schauen wir uns daher erst einmal den Prozessor und seine Sprache an. Hier gibt es keine Variablen oder solche komfortablen Befehle wie PRINT 16^3. Der Programmierer hat zwar etliche Befehle zur Verfügung, diese beziehen sich jedoch nur auf Bits, Bytes, Worte, Langworte und Register. Will man einen Wert ablegen, so muß man ihn sich in eine Speicherzelle des Arbeitsspeichers schreiben und sich dessen Adresse auch noch genau merken...

Doch keine Angst, so kompliziert, wie das alles klingt, ist es gar nicht. Betrachten wir ein Beispiel und nehmen wir an, in den Speicherzellen $1000 und $1002 liegen zwei Datenworte, die addiert werden sollen. Das Ergebnis soll dann nach $1004 gebracht werden. Ein Maschinenprogramm sähe dafür folgendermaßen aus:

```
MOVE.W $1000,D0
ADD.W  D0,$1002
MOVE.W D0,$1004
```

Das war's. Der Befehl MOVE bewegt, wie der Name sagt, Daten im Rechner. Der Zusatz .W gibt dabei an, daß es sich hier um ein Wort handelt, also 16 Bit. Ebenso könnte dort .B für ein Byte und .L für ein Langwort stehen. Der erste Befehl unseres Programmes nimmt also das Wort aus der Speicherzelle $1000 und legt es in D0. Dieses D0 ist ein Datenregister, ein 32-Bit-Speicher in der CPU selbst. Der MC 68000 besitzt derer 8, D0 bis D7. In diesen Registern können Daten gespeichert werden, die man gerade verarbeiten will. Sie haben dem Arbeitsspeicher gegenüber noch weitere Vorteile, auf die wir aber erst einmal nicht eingehen wollen.

Das Datenregister 0 enthält nun also den Wert der Speicherzelle $1000. Nun wird mit dem ADD.W dieser Wert mit dem Inhalt von $1002 addiert. Auch hier steht der Zusatz .W für die aktuelle Datengröße, eben dem Wort. Das Ergebnis der Addition befindet sich nun in D0, von wo aus es nur noch in die gewünschte Speicheradresse gebracht werden muß. Dafür nehmen wir wieder den MOVE.W-Befehl, dessen Anwendung hier fast dieselbe ist wie eben. Die MOVE-Anweisung bringt immer den Wert des zuerst angegebenen Speichers (oder Registers) in den zweiten. Eine allgemeine Schreibweise dafür wäre:

MOVE.Datentyp von , nach

Mit Datentyp ist der Buchstabe .B, .W oder .L gemeint, "von" und "nach" gibt an, woher und wohin die Daten kommen sollen. Für diese Adressierungen bietet der MC 68000 eine große Auswahl. In folgender Tabelle sind diese Adressierungsarten zusammengefaßt:

Adressierungsart	Beispiel
Register direkt:	
- Datenregister direkt	CLR D0
- Adressregister direkt	MOVEA A1,A6
- Statusregister direkt	MOVE D0,SR
- Bedingungsregister direkt	MOVE D0,CCR

Daten unmittelbar:	
- unmittelbar lang	MOVE #$20000,D0
- unmittelbar kurz	MOVE #$20,D0
- unmittelbar schnell	MOVEQ #9,D0
absolut:	
- absolut kurz	NOT $2000
- absolut lang	NOT $18000
Adress-Register indirekt:	
- einfach	CLR (A0)
- mit Distanz	CLR 80(A0)
- mit Distanz und Index	MOVE 8(A0,D0),$400
- mit Predecrement	CLR -(A7)
- mit Postincrement	CLR (A7)+
Programmzähler relativ:	
- mit Adressdistanz	MOVE 4(PC),A0
- mit Distanz und Index	MOVE 8(PC,A0),D0
- Branch-Befehl	BRA Label

Wie aus obiger Tabelle hervorgeht, unterscheiden sich die Adressierungsarten in 5 Kategorien. Die ersten 3 davon sind leicht zu durchschauen:

Register direkt: hier wird als Quelle oder Ziel direkt ein Register verwendet, dessen Inhalt verschoben bzw. verändert wird. Die möglichen Register sind die Datenregister D0 bis D7, die Adressregister A0-A7, das Statusregister SR und das Conditioncode-Register CCR. Bei A7 ist Vorsicht geboten, da dieses Adressregister den Stackpointer darstellt.

Daten unmittelbar: dabei werden genau die Daten angegeben, die gewünscht werden. Natürlich ist diese Adressierungsart nur als Quelle sinnvoll, denn wie soll der Prozessor eine Konstante verändern können? Diese Adressierung entspricht ungefähr dem POKE-Befehl, da MOVE #10,1000 wie POKE 1000,10 einen festen Wert (10) in eine Speicherzelle schreibt.

Absolut: es wird eine direkt angegebene Adresse angesprochen und ihr Inhalt verwendet. Mit PEEK und POKE können ebenso nur absolute Adressen angesprochen werden. Es gibt aber noch mehr Arten, eine bestimmte Speicherzelle auszuwählen. Diese Adressierungsarten sind zwar teilweise etwas verschlungen, aber sehr wirkungsvoll.

Nun sehen wir uns die anderen zwei Arten der Adressierung an. Zuerst zur indirekten Adressierung:

Hierbei wird nicht das verwendete Register selbst angesprochen, sondern sein Inhalt als Adresse interpretiert. Mit dieser Methode können sehr einfach Zeiger verarbeitet werden, indem man den Wert des Zeigers in ein Register läd und damit indirekt adressiert. Ein Beispiel:

```
MOVE.L 1102,A0        * Bildschirmadresse in A0
MOVE.W #1,(A0)        * und Punkt ins Bild setzen
```

Dieses winzige Maschinenprogramm ist mit BASIC nur durch einen recht hohen Aufwand nachzubilden. Ein entsprechendes BASIC-Programm sieht dann so aus:

```
10   defdbl a,b        : rem Langwörter definieren
20   b = 1102          : rem b als Adresse des Zeigers
30   a = peek(b)       : rem Bildschirmadresse ermitteln
40   poke a,1          : rem und Punkt setzen.
```

Schon an diesem kleinen Beispiel wird der Vorteil der indirekten Adressierung klar. Nun kann zusätzlich zur Adresse selbst noch eine Distanz angegeben werden, ein sogenannter Offset. Der Prozessor addiert dann zu der im Register stehenden Adresse den Offset und nimmt das Ergebnis erst als effektive Adresse. In obigem Beispiel kann dies verwendet werden, um nicht das erste Speicherwort des Bildschirms, sondern das n-te zu verändern bzw. zu lesen. Es hieße dann MOVE.W #1,n(A0), wobei n den Offset darstellt. Soll der Punkt z.B. in die zweite Zeile des Bildschirms, so nimmt man MOVE.W #1,80(A0). Die Distanz ist allerdings nur in Bytes anzugeben und darf maximal 32767

betragen. Somit kann der gesamte Bildschirmspeicher bearbeitet werden.

Der Offset ist nun aber eine Konstante. Will man im Programm diese Distanz variieren, so kann man zusätzlich noch ein Register angeben, dessen Inhalt dann ebenfalls zur Adresse addiert wird. Der konstante Offset darf dabei nur noch maximal 127 sein, das Register allerdings kann auch ein Langwort beinhalten. Dieses Wort oder Langwort ist vorzeichenbehaftet, d.h. sein höchstes Bit gibt das Vorzeichen des Wertes an (1 = negativ, 0 = positiv). Durch diese Adressierungsart kann der gesamte Speicherbereich des ATARI ST adressiert werden, was allerdings nicht unbedingt nötig ist. Interessant ist sie, wenn große Datenmengen verarbeitet werden, deren Position im Speicher variabel ist.

Nun zu den indirekten Adressierungen mit Postinkrement und Predekrement. Diese eigenartig klingenden Wörter bedeuten, daß das gewählte Register vor (post) oder nach (pre) dem Zugriff erhöht (inkrementiert) bzw. erniedrigt (dekrementiert) wird. Diese Art ist enorm wichtig und kommt eigentlich in jedem Programm vor. So können nämlich Stack-Operationen sehr leicht durchgeführt werden. Der Stack stellt ja eine Aneinanderreihung von Daten und Adressen dar, auf die oft zugegriffen wird, wie z.B. beim RTS-Befehl (vergl. RETURN in BASIC), der sich die Rücksprungadresse zum Hauptprogramm vom Stack holt. Programmierer von 8-Bit-Rechnern wie dem 6502, der in den ATARI XLs oder C64ern eingesetzt ist, kennen diesen Zugriff als PLA oder PHA. Diese Befehle können aber nur immer ein Byte vom Stapel nehmen bzw. drauflegen.

Dafür werden nun im ATARI ST die Postinkrement- und Predekrement-Adressierungen im 68000 verwendet, die zusätzlich noch Wort- und Langwortverarbeitung ermöglichen. Soll ein Wort vom Stapel genommen werden, so lautet der Befehl hierfür MOVE.W (SP)+,D0. Das Wort steht dann in dem Datenregister D0, und der Stapelzeiger SP (Stack-Pointer) wird um ein Wort erhöht. Wird derselbe Befehl noch einmal ausgeführt, so erhält man schon den nächsten Wert aus dem Stapel, ohne daß eine Veränderung des Stapelzeigers nötig wäre.

Das gleiche gilt für das Ablegen von Daten auf dem Stack. Da aber der Stack-Pointer immer auf das zuletzt abgelegte Datum zeigt, muß vor dem Hinzufügen des nächsten Datums der Zeiger um die der Länge des Datums entsprechenden Bytes erniedrigt werden. Dafür wird dann die Predekrement-Adressierung verwendet. Ein Langwort auf den Stack abzulegen kann z.B. mit MOVE.L #$F8000,-(SP) erledigt werden. Der Stackpointer ist danach um 4 Bytes erniedrigt.

Die letzte Adressierungsart lautet: Programmzähler relativ. Die Adressierung kann damit in Abhängigkeit vom Programmzählerstand erfolgen, der die momentane Adresse des bearbeiteten Befehls enthält. Somit ist es möglich, Tabellen in einem Programm anzulegen und darauf zuzugreifen, obwohl die wirkliche Adresse des Programms im Arbeitsspeicher des Computers variabel und damit unbekannt ist. Diese Möglichkeit der Adressierung befähigt den 68000-Prozessor, auch große Programme irgendwo im Speicher zu verarbeiten. Dies vereinfacht auch die Kombination von BASIC und Maschinensprache, da man solcherart gestaltete Programme z.B. in Textvariablen ablegen und laufen lassen kann. Diese sehr bequeme und praktische Methode ist in Computern ohne eine solche Adressierungsart immer mit einigen Schwierigkeiten verbunden, wenn das Programm auf eigene Daten zugreifen muß und nicht weiß, wo es selbst steht. Die Schreibweise der Befehle in programmzählerrelativen Adressierungen ist der indirekten Register-Adressierung sehr ähnlich. Hier wird lediglich statt des Registers PC (Program-Counter) eingesetzt.

Eine Spielart der relativen Adressierung stellt noch der Branch-Befehl dar. Hier wird nicht auf Daten zugegriffen, sondern der Programmablauf selbst gesteuert. Durch einen Branch-Befehl wird das Programm verzweigt (Branch = Zweig), wobei dies unbedingt durch eine BRA-Anweisung (Branch Always) oder bedingt durchgeführt wird. Diese Bedingungen sind recht vielfältig wählbar. Meist beziehen sie sich auf einen vorhergehenden Vergleich zweier Daten. Das sieht z.B. so aus:

```
CMP.W  #10,D0
BLT    KLEINER
...
```

Hier wird zu dem Programmabschnitt LABEL verzweigt, wenn der Inhalt des Registers D0 kleiner als 10 ist. BLT bedeutet ausgesprochen "Branch if Less Than" und verdeutlicht seine Funktion somit schon durch seine Schreibweise. Nachfolgende Tabelle enthält alle Branch-Befehlsvariationen:

Mnemonic	heißt Branch if...	auf Deutsch	springt, wenn
BEQ	equal	gleich	Z=1
BNE	not equal	ungleich	Z=0
BPL	plus	positiv	N=0
BMI	minus	negativ	N=1
BGT	greater than	größer	Z+(N&V)=0
BLT	less than	kleiner	N&V=1
BGE	greater or equal	größer gleich	N&V=0
BLE	less or equal	kleiner gleich	Z+(N&V)=1
BHI	higher	höher	C+Z=0
BLS	lower or same	niedriger gleich	C+Z=1
BCS	Carry set	Carry gesetzt	C=1
BCC	Carry clear	Carry gelöscht	C=0
BVS	overflow	Überlauf	V=1
BVC	no overflow	kein Überlauf	V=0
BRA	branch always	springe immer	
BSR	branch to subroutine	Unterprogrammsprung	

Das Plus-Zeichen bedeutet hierbei logisches ODER, & steht für logisch UND (siehe Abschnitt 3). Die rechts angeführten Ziffern stehen jeweils für ein Flag-Bit im Condition-Code Register (CCR). Jedes dieser Bits hat eine eigene Bedeutung und wird bei verschiedenen Operationen beeinflußt.

Wo wir gerade bei den Verzweigungen und damit bei Programmablauf-Steuerungenind, können wir uns gleich einmal alle Programmsteuer-Befehle ansehen. Einige davon verzweigen in Programmteile, nach deren Bearbeitung das Hauptprogramm an der Stelle weiterläuft, wo die Verzweigung stattgefunden hat.

Davon kennen wir ja eigentlich schon den JSR bzw. BSR-Befehl, da JSR (Jump to Sub-Routine) dem BASIC-Befehl GOSUB sehr ähnlich ist. Auch der JMP (Jump)-Befehl ist uns als GOTO bekannt. Doch der Prozessor kennt noch mehr Verzweigungen dieser Art.

Da wäre z.B. der TRAP-Befehl, der gerade im ATARI ST eine sehr große Rolle spielt. Mit dieser Anweisung wird das GEMDOS oder das BIOS aktiviert, je nach der Nummer, die dem TRAP folgt. TRAP #1 ruft das GEMDOS auf, TRAP #13 das BIOS und TRAP #14 das erweiterte BIOS, das XBIOS. Einen solchen Aufruf einer Funktion nennt man auch Exception oder Software-Interrupt. Die Funktion, die das aufgerufene Programm ausführen soll, wird dabei auf dem Stack als Funktionsnummer übergeben, ebenso wie eventuelle Parameter. Nach Beendigung der entsprechenden Routine im Betriebssystem bekommt das aufrufende Programm wieder die Kontrolle über den Rechner. Dieses muß dann alle Parameter, die es übergeben hat, wieder vom Stack holen, damit dieser wieder aktuell ist. Das wird im allgemeinen durch eine Addition mit dem Stapelzeiger SP (Stack Pointer) erledigt.

Hier nun eine Liste der Programmsteuerbefehle des MC 68000:

Mnemonic	Bedeutung
Bcc	Verzweige bedingt
BRA	Verzweige unbedingt
BSR	Verzweige an ein Unterprogramm
CHK	Prüfe ein Datenregister auf Grenzen
DBcc	Prüfe Bedingung, dekrementiere und verzweige
JMP	Springe zur Adresse
JSR	Springe an ein Unterprogramm
NOP	Keine Funktion (No Operation)
RESET	Peripherie zurücksetzen
RTE	Rückkehr aus einer Exception
RTR	Rückkehr mit Laden der Flags
RTS	Rückkehr aus einem Unterprogramm
Scc	Setze ein Byte abhängig von einer Bedingung

STOP	Verarbeitung anhalten
TRAP	Springe in eine Exception
TRAPV	Prüfe, ob Flag gesetzt, dann evtl. Exception

In dem kleinen Beispielprogramm, in dem wir zwei Zahlen addiert haben, wurde eine arithmetische Operation, eben die Addition, durchgeführt. Dem gegenüber steht natürlich auch die Möglichkeit der Subtraktion. In diesen beide Operationen erschöpft sich die Rechenfähigkeit eines üblichen 8-Bit-Prozessors wie dem 6502. Der MC 68000 kann aber noch mehr. Er kann zusätzlich noch multiplizieren, dividieren und all diese Operationen mit verschiedenen Datengrößen durchführen.

Hier nun eine Übersicht über die arithmetischen Operationen mit ganzen Zahlen:

Mnemonic	Bedeutung
ADD	Binäre Addition
ADDA	Binäre Addition von Adressen
ADDI	Addition mit einer Konstanten
ADDQ	Schnelle Addition einer Konstanten
ADDX	Addition mit Übertrag
CLR	Löschen eines Operanden
CMP	Vergleich zweier Operanden
CMPA	Vergleich zweier Adressen
CMPI	Vergleich mit einer Konstanten
CMPM	Vergleich zweier Speicheroperanden
DIVS	Division mit Vorzeichen
DIVU	Division ohne Vorzeichen
EXT	Vorzeichenrichtige Erweiterung
MULS	Multiplikation mit Vorzeichen
MULU	Multiplikation ohne Vorzeichen
NEG	Negation eines Operanden (Zweierkomplement)
NEGX	Negation eines Operanden mit Übertrag
SUB	Binäre Subtraktion
SUBA	Binäre Subtraktion von Adressen
SUBI	Subtraktion einer Konstanten

SUBQ	Schnelle Subtraktion einer Konstanten
SUBX	Subtraktion mit Übertrag
TST	Testen eines Operanden auf Null

Zusätzlich zu der Verarbeitung von ganzen Zahlen kann der Prozessor auch direkt mit BCD-Zahlen operieren. Dafür gibt es folgende Befehle:

Mnemonic	Bedeutung
ABCD	Addition zweier BCD-Zahlen
NBCD	Negation einer BCD-Zahl (Neunerkomplement)
SBCD	Subtraktion zweier BCD-Zahlen

Als nächstes nun die logischen Operationen, die wir teilweise auch vom BASIC her kennen:

Mnemonic	Bedeutung
AND	Logisch UND
ANDI	Logisch UND mit einer Konstanten
EOR	Exklusiv-ODER
EORI	Exklusiv-ODER mit einer Konstanten
NOT	Inversion eines Operanden
OR	Logisch ODER
ORI	Logisch ODER mit einer Konstanten
TAS	Prüfe und setze ein spezielles Bit

Auch einzelne Bits können direkt manipuliert werden. Dafür stehen folgende Befehle zur Verfügung:

Mnemonic	Bedeutung
BCHG	Verändere ein bestimmtes Bit
BCLR	Lösche ein bestimmtes Bit
BSET	Setze ein bestimmtes Bit
BTST	Teste ein bestimmtes Bit

Des weiteren kann der Prozessor einen Operanden noch in sich verschieben und rotieren:

Mnemonic	Bedeutung
ASL	Arithmetische Verschiebung nach links (*2)
ASR	Arithmetische Verschiebung nach rechts (/2)
LSL	Logische Verschiebung links
LSR	Logische Verschiebung rechts
ROL	Rotation nach links
ROR	Rotation nach rechts
ROXL	Rotation nach links mit Übertrag
ROXR	Rotation nach rechts mit Übertrag

Und nun kommen wir zu den Befehlen, die Daten im Rechner bewegen:

Mnemonic	Bedeutung
EXG	Austauschen zweier Registerinhalte
LEA	Lade eine effektive Adresse
LINK	Baue Stackbereich auf
MOVE	Übertrage ein Datum
MOVE from SR	Übertrage den Statusregister-Inhalt
MOVE to CCR	Flags laden
MOVE to SR	Statusregister laden
MOVE USP	User-Stackpointer laden
MOVEA	Übertrage eine Adresse
MOVEM	Übertrage mehrere Register
MOVEP	Übertrage Daten zur Peripherie
MOVEQ	Übertrage schnell eine Konstante
PEA	Lege eine Adresse auf dem Stack ab
SWAP	Vertausche die Registerhälften
UNLK	Baue Stackbereich ab

Das waren alle Befehle des MC 68000. Durch die Kombination mit den diversen Adressierungsarten lassen sich sehr viele Befehle zusammenstellen, um ein Programm so effizient wie möglich zu machen.

Nun wollen wir uns einmal das Programm ansehen, das wir zur Umschaltung des RGB-Bildes auf 50 Hertz verwendet haben. Das Programm sieht im Maschinencode, auch Mnemonik genannt, so aus:

```
CLR.L   -(SP)
MOVE.W #$20,-(SP)
TRAP   #1              * Set Supervisor-State
ADDQ.L #6,A7           * Stack reparieren
MOVE.B #2,$FF820A      * PAL-Frequenz setzen
MOVE.L D0,-(A7)        * alter SSP
MOVE.W #$20,-(SP)
TRAP   #1              * Set User-State
ADDQ.L #6,SP           * Stack reparieren
CLR.W  -(SP)
TRAP   #1              * zurück ins Desktop
```

Der erste TRAP bewirkt, daß das GEMDOS das System in den Supervisor-Modus versetzt. Das ist hier erforderlich, da der Zugriff auf den I/O-Bereich privilegiert ist und vom User-Modus aus einen Bus-Error erzeugen würde (und zwei Rauchpilze...). Nach der Korrektur des Stackpointers wird dann die Umschaltung auf PAL-Frequenz von 50 Hz selbst vorgenommen. Danach wird wieder das GEMDOS mit TRAP #1 und derselben Funktionsnummer wie vorhin ($20) aufgerufen. Diesmal wird der alte Supervisor-Stackpointer, der vom ersten Aufruf in D0 übergeben worden ist, an die Funktion mit übergeben. Danach befindet sich der Rechner wieder im Status des Users. Nach einer weiteren Reparatur des Stacks wird dann die GEMDOS-Funktion 0 aufgerufen, die zum aufrufenden Programm zurückgibt.

Das BASIC-Programm zur Generierung dieses Maschinenprogramms enthält noch einige zusätzliche Werte, die in dem erzeugten Applikationsprogramm voranstehen. Diese Werte gehören nicht zum Programm selbst, aber sie werden vom GEMDOS benötigt für die Erkennung der Länge und Art des Programmes. Wenn wir ein Maschinenprogramm von BASIC aus

mit CALL aufrufen, so brauchen wir diesen Vorspann nicht. Ebenfalls unnötig, sogar falsch, wäre die Verwendung des TRAP #1-Aufrufes mit der Funktionsnummer 0, da das GEMDOS daraufhin nicht wieder an das BASIC-Programm, sondern an das Desktop übergibt. Ein solches Programm wird im allgemeinen mit dem RTS-Befehl abgeschlossen, der die Kontrolle wirklich dem BASIC zurückgibt.

6.4.1 Kombination mit BASIC

Die Vorteile der Maschinensprache sind also so groß, daß es sicher interessant wäre, sie mit der wesentlich einfacher zu schreibenden Programmiersprache BASIC zu kombinieren. Hierfür existieren auch einige Befehle in BASIC, von denen wir PEEK und POKE ja schon kennen. Doch fehlt uns noch die Möglichkeit, Maschinenprogramme aus einem BASIC-Programm aufzurufen. Dazu ist der CALL-Befehl vorgesehen. Dieser Maschinen-Aufruf bietet zusätzlich noch die Möglichkeit Parameter an das Maschinen-Programm zu übergeben, was eine sehr effiziente Programmierung ermöglicht.

Betrachten wir erst einmal den Aufruf selbst. Hierfür wollen wir uns ein Beispiel ansehen, welches es ermöglicht, die Baud-Rate der seriellen Schnittstelle direkt aus einem BASIC-Programm heraus einzustellen. In dem Abschnitt über die Einstellung der Schnittstellen haben wir ein ähnliches Programm kennengelernt, welches aber nur vom Desktop aus aufrufbar war. Nun sehen wir uns die andere Version an:

```
10   rem *** Baudrate konfigurieren ***
12   a$=space$(40)              : rem Speicher reservieren
14   b=peek(varptr(a$)+2)       : rem Adresse feststellen
20   for i=0 to 36 step 2       : rem in einer Schleife
30   read a                     : rem Daten einlesen
40   poke b+i,a                 : rem und einsetzen
50   next i
60   input "Baudraten-code ";x  : rem Code eingeben
70   poke b+22,x                : rem einfügen
```

```
80  call b                      : rem und aufrufen
90  end                         : rem das war's !
100 rem  -- Maschinen-Programm --
110 data 16188,-1,16188,-1,16188,-1
120 data 16188,-1,16188,-1,16188,15
130 data 16188,15,20046,-8196,0,14
140 data 20085
```

Wie man sieht, ist das Maschinenprogramm selbst wesentlich kürzer als das vorher vorgestellte. Das liegt an der Programm-parameter-Tabelle, die vor jedem Applikationsprogramm liegen muß, damit das GEMDOS die Anfangs- und Endadresse des Programmes festlegen kann. Außerdem ist die Beendigung eines Programmes über einen speziellen GEMDOS-Aufruf nötig, wenn die Kontrolle wieder an das Desktop bzw. das Command-Programm übergeben werden soll.

Beim Aufruf mit dem CALL-Befehl ist dagegen kein Parameter-Vorspann nötig, ebenso wie ein einfacher RTS-Befehl des Maschinenprogrammes (RTS entspricht dem BASIC-Befehl RETURN) zur Rückkehr in das aufrufende BASIC-Programm genügt.

Das Maschinenprogramm wird wieder aus DATA-Zeilen gelesen und in den Speicher gepoket. Die Besonderheit in diesem Beispiel-Programm ist die, daß dieser Speicher beliebig verschiebbar ist. Aus diesem Grunde kann die Textvariable A$, deren Speicheradresse ja irgendwo liegen kann, zur Aufnahme des Maschinenprogrammes dienen. Dies birgt einen großen Vorteil. Auf diese Weise können mehrere Maschinenprogramme in einem BASIC-Programm integriert sein, ohne daß man sich Gedanken über die Speicherstruktur machen muß. Die Verwendung irgendeines festen Speicherplatzes kann Probleme ergeben, wenn das GEM oder GEMDOS diesen Adressbereich ebenfalls benutzen und das Maschinenprogramm zerstören. Ein Aufruf eines solchen feststehenden Programmes führt dann zum Absturz.

In der Variablen A$ liegt dann also nach Beendigung der FOR-NEXT-Schleife unser Maschinenprogramm. Als nächstes wird nach dem Code für die Baud-Rate gefragt, da ja jede beliebige Geschwindigkeit eingestellt werden kann. Beliebig ist zwar übertrieben, aber die 16 möglichen Baudraten reichen normalerweise völlig aus, um alle Bedürfnisse an Geschwindigkeit der Datenübertragung abzudecken. Man gibt also die Kennzahl der gewünschten Baudrate ein, die von 0 (19200 Baud) bis 15 (50 Baud) reicht. Eine vollständige Tabelle finden Sie in Kapitel 5.1 dieses Buches.

Der eingegebene Wert wird nun in das Maschinenprogramm eingesetzt, indem er in die entsprechende Position gepoket wird. Danach erfolgt der eigentliche Aufruf des Maschinenprogrammes mit CALL B, wobei B die Adresse der Variablen und damit des Programmes ist. Das Programm ruft dann durch einen TRAP-Befehl das erweiterte BIOS auf und übergibt die Parameter der Schnittstellen-Einstellungen. Dieses Maschinenprogramm sieht folgendermaßen aus:

```
* Einstellung der Baud-Rate *

MOVE.W #-1,-(SP)      * Synchron - Charakter
MOVE.W #-1,-(SP)      * Transmitter Status Register
MOVE.W #-1,-(SP)      * Receiver Status Register
MOVE.W #-1,-(SP)      * USART Control Register
MOVE.W #-1,-(SP)      * XON/XOFF und RTS/CTS
MOVE.W #15,-(SP)      * Baudrate (BSPl. 50 Baud)
MOVE.W #15,-(SP)      * Kommando

TRAP   #14            * XBIOS - Aufruf
ADD.L  #14,SP         * Stack korrigieren

RTS                   * Rückkehr zum BASIC
```

Die 15, die in der 6. Zeile liegt und die Baudrate bestimmt, ist die Zahl, die wir mit dem POKE-Befehl verändern. Zusätzlich kann durch ein weiteres POKE auch das Übertragungsprotokoll verändert werden, das mit XON/XOFF und RTS/CTS angegeben ist. Die Adresse dessen wäre in obigen BASIC-Programm POKE B+18,n, um ein neues Protokoll einzustellen. Eine -1 als Parameter bedeutet immer, daß die aktuelle Einstellung übernommen wird. Wenn Sie also mit dem Menüpunkt Schnittstelle konfigurieren RTS/CTS eingestellt haben, wird das auch noch nach der neuen Wahl der Baudrate mit obigem Programm gelten.

Dieses Beispiel demonstrierte den CALL-Befehl als Aufruf eines Maschinenprogrammes. Die mehrmalige Verwendung des CALL B setzt voraus, daß die A$-Variable noch das Programm enthält und das B dessen Adresse. Nun können aber durch BASIC-Programmänderungen diese Adressen der Variablen geändert werden. Deshalb ist es ratsam die Bestimmung der Variablen B vor jedem Aufruf neu vorzunehmen.

Nun wollen wir sehen, wie Parameter an ein Maschinenprogramm übergeben werden können, ohne daß wir immer mit POKE arbeiten zu müssen. Der CALL-Befehl erlaubt die Angabe von Übergabe-Werten in Klammern, z.B. CALL X (1,2,3). Das Maschinenprogramm muß diese Werte nun übernehmen. Wegen akutem Unterlagenmangel mußte ich ein Beispielprogramm schreiben, was mir diese Werte findet und zurückgibt. Dieses Programm ist zwar nicht besonders anwendbar, zeigt aber den Ablauf der Parameterübergabe recht deutlich. Die Werte, die in Klammern stehen, werden auf dem Stack übergeben, deren Anzahl findet das Maschinenprogramm im Datenregister D0. Dazu nun das Programm:

BASIC-Anteil:

```
10   rem *** Calltest ***
20   a$=space$(100)             : rem Platz schaffen
30   a=peek(varptr(a$)+2)       : rem Adresse feststellen
40   bload "calltest.prg",a     : rem Maschinenprogramm laden
90   end                        : rem Ende erster Teil.
```

```
100  call a (1,2,3)          : rem Testaufruf
130  n=peek(8192)            : rem Feststellung der Anzahl
140  print n;" Argumente :"
150  defdbl j                : rem Langwort einstellen
155  d=1                     : rem Zähler initialisieren
160  for j=8208 to 8204+n*4 step 4
170  print d;". : ";peek(j)  : rem Ausgabe des Parameters
180  d=d+1                   : rem Zähler erhöhen
190  next j                  : rem und weiter
```

Maschinen-Anteil:

```
MOVE.W D0,$2000          * Anzahl ablegen
MOVE.L 14(SP),$2010      * erster Parameter
MOVE.L 18(SP),$2014      * zweiter   "
MOVE.L 22(SP),$2018      * dritter   "
MOVE.L 26(SP),$201C      * vierter   "
MOVE.L 30(SP),$2020      * fünfter   " , reicht

RTS                      * und zurück zum BASIC
```

Diese Möglichkeit der Parameterübergabe läßt sich sehr vielfältig anwenden. Ein allgemeines Beispiel dafür soll nun besprochen werden, und zwar die Manipulation von Speicherzellen, deren Zugriff eigentlich nur im privilegierten Status des Programmes erlaubt ist. Der BASIC-Programmierer besitzt für den Rechner nur den Status eines Users, eines Benutzers. Somit erlaubt der Prozessor ihm nur unprivilegierte Speicherzugriffe vorzunehmen. Der Vorteil dabei liegt darin, daß ein normales Programm nicht in die Intimsphäre des Rechners eingreifen kann. Das System ist somit in der Lage, sich selbst vor unautorisierten Manipulationen zu schützen.

Dies hat allerdings einen Haken. Ein BASIC-Zugriff mittels POKE auf einen solchermaßen geschützten Speicherbereich hat unmittelbar einen Absturz zur Folge, so daß sich der Programmierer im Desktop wiederfindet und seine BASIC-Daten weg sind. Das ist um so schlimmer, als daß einige dieser privilegier-

ten Speicherzellen interessante Möglichkeiten der Programmierung eröffnen. Ein Beispiel hierfür ist der I/O-Bereich, durch dessen Manipulation die Programmierung der Sonderbausteine des ATARI ST möglich wird.

```
10   rem *** privilegierter Zugriff ***
20   a$=space$(36)             : rem Platz reservieren
30   b=peek(varptr(a$)+2)      : rem Adresse feststellen
35   b=b+(b mod 2)             : rem gerade Adresse setzen
40   for i=0 to 34 step 2
50   read a                    : rem Daten lesen
60   poke b+i,a                : rem und einschreiben
80   next i
90   rem --- Maschinenprogramm - Daten ---
100  data -20356,2,26140,17063,16188
110  data 32,20033,23695,8815,14,12975
120  data 20,12032,16188,32,20033,23695,20085
```

Dieses Programm erstellt ein Maschinenprogramm, welches einen direkten Zugriff bzw. eine Veränderung eines privilegierten Speichers erlaubt. Es wird aufgerufen mit dem Befehl CALL B (Adresse, Wert), wobei die Adresse ein Langwort und das einzusetzende Wert ein Datenwort ist. Um die Funktion des Programmes zu überprüfen, kann man durch CALL B (491520, 21845) beim 512 KByte-ST bzw. durch CALL B (1015808, 21845) für die 1 MByte-Version einen kleinen Strich an die Bildoberkante schreiben. Dies ist zwar auch mit einem POKE möglich, aber es ist ja nur ein Test.

Zum Ausprobieren eines privilegierten Zugriffes kann man z.B. die Vertikalfrequenz des Farbbildes ändern, was ja einen Zugriff auf den zuständigen Video-Chip erfordert. Die Adresse dafür ist $FF820A, also 16744970, und soll für PAL-Fernseher (50 Hz) den Wert 2 enthalten. Um diesen Wert zu verändern, muß man allerdings den gewünschten Wert mit 256 multiplizieren, da ja ein Wort eingesetzt wird. Die Adresse soll aber nur ein Byte bekommen, so daß das niederwertigere Byte des Wortes über das Ziel hinausschießt. Durch die Multiplikation mit dem

oben genannten Wert, der $100 entspricht, wird das eigentliche Byte an die richtige Stelle gerückt.

Sehen wir uns nun das Maschinenprogramm an, welches wir in den String A$ gelegt haben.

```
Source File: TEST.S

 1                           * Privileg-Umgehung *
 2
 3 000000 B07C0002           CMP.W  #2,D0       * 2 Argumente ?
 4 000004 661C               BNE    EXIT        * nein => Exit
 5
 6 000006 42A7               CLR.L  -(SP)
 7 000008 3F3C0020           MOVE.W #$20,-(SP)
 8 00000C 4E41               TRAP   #1          * Supervisor !
 9 00000E 5C8F               ADD.L  #6,SP       * Stack rep.
10
11 000010 226F000E           MOVE.L 14(SP),A1 * Adresse holen
12 000014 32AF0012           MOVE.W 20(SP),(A1) * und setzen
13
14 000018 2F00               MOVE.L D0,-(SP)
15 00001A 3F3C0020           MOVE.W #$20,-(SP)
16 00001E 4E41               TRAP #1            * User-State
17 000020 5C8F               ADD.L #6,SP        * Stack rep.
18
19 000022 4E75         EXIT: RTS                * und zurück !
```

Bevor das Programm richtig anfängt, wird getestet, ob der CALL-Aufruf 2 Parameter übergeben hat. Wenn nicht, wird direkt verzweigt zum RTS, wo die Kontrolle wieder an das BASIC zurückgegeben wird. Ist alles in Ordnung, so versetzt der TRAP-Befehl mit diesen auf dem Stack übergebenen Argumenten das System in den Supervisor-Modus, wo privilegiert gearbeitet wird. Nach dem TRAP muß, wie in allen Fällen, der Stack wieder repariert werden, da die in ihm übergebenen Argumente nicht mehr benötigt werden.

Als nächsten Schritt lädt das Programm die übergebene Adresse in das Adressregister A1, um dann den zu schreibenden Wert indirekt dorthin zu legen. Das war alles. Nun muß nur noch das System wieder in den normalen User-Status gebracht werden, was das nächste TRAP erledigt. Danach wird mit dem RTS-Befehl in das aufrufende Programm, eben das BASIC, zurückgesprungen.

Die Form des obigen Maschinenprogramm-Listings ist diejenige, die der Assembler des Entwicklungspaketes erstellt, wenn dem Aufruf des Assemblers ein >TEST.ASM nachgestellt wird. Dieser willkürliche Programmname wird dann als Ausgabedatei für das entstandene Listing aufgefaßt, so daß obige Form in dieses File abgelegt wird. In dieser Datei stehen noch einige zusätzliche Informationen, wie Fehleranzahl und Label-Liste, die hier jedoch unwichtig sind. Interessant sind noch die entstandenen Hexadezimal-Zahlen, deren Dezimalwerte in den DATA-Zeilen unseres BASIC-Programmes auftauchen.

7. Programmierung (am Beispiel BASIC)

Nach all diesen Betrachtungen des ATARI ST von innen und außen wollen wir uns nun der Programmierung des Rechners zuwenden. Da die wohl beliebteste Sprache unter den Hobby-Programmierern BASIC ist, nehmen wir diese Sprache für unsere Anwendungen. BASIC hat auch den Vorteil, daß die Programme leicht zu überschauen und zu verstehen ist, so daß Sie die Beispiele, die diesem Abschnitt folgen werden, leicht für Ihre Zwecke umschreiben können. Einige Programme sind Ihnen bei der Lektüre dieses Buches begegnet, die in dieser Sprache geschrieben sind, und vielleicht haben Sie diese oder jene Änderung daran bereits vorgenommen. Solche Manipulationen sind gerade in BASIC meist unkritisch, da schlimmstenfalls eine Fehlermeldung erfolgen kann.

Doch gerade das ist in den Programmen, in denen PEEK und POKE vorkommen, nicht mehr so sicher. Schließlich nimmt man bei dem Einsatz dieser Befehle, besonders POKE, einen Eingriff in die Maschinenebene vor. Ein einziges falsches POKE kann dabei leicht ins Auge gehen. Bevor Sie also mit diesen Befehlen experimentieren, sollten Sie sich besser vergewissern, daß bei einem Absturz nichts verloren geht. Daher ist es ratsam, abgetippte Beispielprogramme erst einmal abzuspeichern, denn ein Tippfehler passiert schnell.

Laden wir also nun den BASIC-Interpreter und beginnen den interessantesten Teil der Arbeit mit einem Computer, der Programmierung selbst.

Die vorgestellten Programme in den folgenden Abschnitten enthalten, wie man von einem Peeks- und Pokes-Buch eigentlich erwarten sollte, viele PEEK- und POKE-Befehle. Wer schon einmal mit einem Computer in BASIC gearbeitet hat, wird in der Anwendung dieser Befehle als solches keine Probleme sehen. PEEK liest eine Speicherzelle aus und POKE verändert eine.

Das ist zwar richtig, aber dennoch gibt es einen Punkt, den man unbedingt beachten muß, wenn man einen 16-Bit-Rechner, wie den ATARI ST, mit Personal-Basic vor sich hat. Der Prozessor in diesem Computer arbeitet ja mit Bytes, Worten und Langworten. Das BASIC des ST tut dies in Verbindung mit PEEK und POKE aber auch! Aus diesem Grunde erhält man oft mit einem PEEK-Befehl immens hohe Werte zurück, die sicher nicht in ein Byte passen würden.

Nun ist es aber wichtig, die momentan verwendete Wortbreite genau zu kennen bzw. zu definieren, wenn man mit POKE eine Speicherzelle manipulieren will. Angenommen, wir wollen das Byte an der Adresse 8000 auf 0 setzen. Die Bytes vor und nach dieser Adresse sollen dabei unverändert bleiben, da sie auch irgendeine Bedeutung haben. Wenn wir nun einfach schreiben POKE 8000,0, so wird unser Zielbyte zwar auf 0 gesetzt, das darauffolgende Byte in Speicherzelle 8001 jedoch auch. Das liegt einfach daran, daß der POKE-Befehl normalerweise nur Worte in den Speicher schreibt, so daß der Zugriff auf ausschließlich ein Byte so nicht möglich ist.

Ein anderes Problem ergibt sich, wenn wir einen Zeiger im Speicher ändern wollen. Ein Zeiger ist üblicherweise immer ein Langwort, also 4 Bytes lang. Somit müssen wir eigentlich zwei POKE-Befehle eingeben, um einen solchen Zeiger vollständig zu ändern. Diese Methode hat aber zwei Haken. Erstens muß man die Zahl, die in den Zeiger gelegt werden soll, in ein höher- und ein niederwertiges Wort zerlegen, was eine zusätzliche Rechnung im Programm erfordert. Zweitens wird dabei der Zeiger zwischen den beiden POKE-Befehlen einen zufälligen Gesamtwert haben. Dies kann sich sehr unschön auswirken, wenn der Zeiger von einer Interrupt-Routine verwendet wird, die vielleicht gerade dann darauf zugreift, wenn das eine POKE erfolgt ist und das andere nicht. Somit hat der Zeiger einen undefinierten Zustand und kann je nach Verwendung durch die Interruptroutine zum Absturz des Systems führen.

Diese Problematik ist von den Programmierern des BASIC-Interpreters erkannt worden. Sie haben eine Möglichkeit geschaffen, einen in der Breite definierten Speicherzugriff vorzunehmen.

Die Unterscheidung, ob ein Wort oder ein Langwort zu POKEn ist oder durch PEEK ausgelesen werden soll, wird anhand der Definition der Adressvariablen in dem jeweiligen Befehl vorgenommen. Ist beim Aufruf X = PEEK(A) oder POKE A,X die Adressvariable A vorher im Programm als Variable mit doppelter Genauigkeit definiert, so wird ein Langwort gelesen bzw. geschrieben. Eine solche Definition wird durch die Anweisung DEFDBL A durchgeführt.

Ist diese Definition nicht erfolgt, oder wurde die Variable mit DEFSNG A als einfach genaue Variable eingestellt, so verarbeitet der BASIC-Befehl nur ein Wort.

Nun fehlt noch die Festlegung, daß ein Byte gelesen oder geschrieben wird. Dafür wird ein zusätzlicher Befehl nötig, den uns das BASIC bereitstellt. Er lautet DEF SEG = 1 und bewirkt zweierlei. Einerseits wird ab sofort nur noch mit Bytes gearbeitet, was durch DEF SEG = 0 wieder rückgängig gemacht werden kann. Andererseits wird nicht mehr das adressierte Byte selbst, sondern das darauf folgende Byte gelesen. Somit muß also die wirklich gewünschte Adresse um eins vermindert werden, damit das richtige Byte getroffen wird. Diese umständliche Adressierung hat allerdings auch einen Vorteil. Da der Offset, also die Differenz zwischen angegebener und bearbeiteter Adresse, mit dem DEF SEG = n Befehl eingestellt werden kann (n ist dieser Offset), kann so indiziert adressiert werden. Das heißt, daß die Befehlsfolge DEF SEG = n : X = PEEK(A) die Adresse A+n ausliest. Somit kann bei konstanter Adresse durch Variation des Offsets z.B. ein Datensatz verarbeitet werden, der allerdings erst bei A+1 beginnt. Ein Offset von 0 bewirkt die Umschaltung auf Wortverarbeitung.

Diese Technik der Datenbreiten-Definition erfordert eine sehr genaue Programmierung. Häufig ist mir der Fehler unterlaufen, daß ich eine Umschaltung auf Byteverarbeitung nicht rück-

gängig gemacht habe, so daß das Programm beim ersten Durchlauf zwar funktionierte, beim zweiten Mal jedoch abstürzte. Der RUN-Befehl setzt die Definition nämlich nicht etwa zurück, sondern das muß explizit im Programm vorgenommen werden.

Wenn Sie im Zuge des Ausprobierens der nun folgenden Programme mit den PEEK- und POKE-Befehlen experimentieren, so achten Sie bitte auf die exakte Definition der Datenbreite, die momentan eingestellt ist. Ein völlig richtig eingetipptes Programm kann nicht funktionieren, wenn es Bytes statt Worte verarbeitet. Um ganz sicher zu gehen, ist es somit ratsam, in die erste Zeile eines Programmes, welches PEEK und POKE verwendet, den Befehl DEF SEG = 0 zu setzen.

7.1 Grafik

Der ATARI ST mit seiner großen Bildauflösung und dem GEM im Betriebssystem ist für die grafische Darstellung mehr als gut gerüstet. Das GEM unterstützt nahezu alles, was sich ein Programmierer an grafischen Darstellungen vorstellen kann. Striche ziehen, Kreise und Ellipsen malen, Rechtecke mit runden oder eckigen Kanten zeichnen und sogar das Ausfüllen beliebiger Formen mit beliebigen Mustern sind so einfache Vorgänge, daß der erfahrene Programmierer von Nicht-GEM-Computern ins Träumen kommen kann. Fangen wir gleich damit an, einen ausgefüllten Kreis zu zeichnen.

Dem besagten Programmierer schwebt jetzt ein Programm mit Sinus- und Cosinusfunktionen vor. Doch das brauchen wir nicht. Der Befehl PCIRCLE X,Y,R erledigt das Zeichnen sofort und schnell. Doch hat dieser Befehl eine Einschränkung; er kann seinen Kreis nur in das OUTPUT-Fenster des BASIC-Bildschirms bringen.

Diese Einschränkung gilt nicht mehr, wenn wir für die Grafik-Funktionen das GEM selbst bemühen. Dann können wir an jede beliebige Stelle auf dem Bildschirm jede beliebige Zeichnung bringen und haben außerdem noch wesentlich mehr Möglichkeiten der Gestaltung, als uns das BASIC bieten kann.

7.1.1 Kreise, Ellipsen und Rechtecke

Um nun besagten Kreis in die obere linke Ecke zu zeichnen, müssen wir einen VDISYS-Aufruf machen, wie uns ja bereits aus dem GEM-Kapitel bekannt ist. Den Kreis zeichnet und füllt folgendes Programm:

```
10   rem *** gefüllten Kreis zeichnen ***
20   color 1,0,1,1,1        : rem Füllfarbe
30   poke contrl,11         : rem Rechteck zeichnen
40   poke contrl+2,3        : rem Parameteranzahl
50   poke contrl+6,0
60   poke contrl+10,4       : rem Funktions-ID
70   poke ptsin,100         : rem X-Mitten-Koordinate
80   poke ptsin+2,100       : rem Y-Mitten-Koordinate
90   poke ptsin+4,0         : rem Dummy
100  poke ptsin+6,0         : rem Dummy
110  poke ptsin+8,50        : rem Radius
120  poke ptsin+10,0        : rem Dummy
130  vdisys 0               : rem und Ausführung
```

Die Funktion 11, die von Zeile 30 in das CONTRL(0)-Feld übertragen wird, kann sogar noch viel mehr als nur Kreise zeichnen. Es lassen sich mit dieser einzigen Funktion 10 verschiedene Grafikarten darstellen. Die Information, welche dieser Bilder nun gezeichnet werden soll, enthält die Funktions-ID, die in CONTRL(5) übergeben wird. Die Zuordnung der ID-Nummern zu den Zeichnungen ist folgende:

ID	Zeichnung
1	gefülltes Rechteck
2	Kreis-Ausschnitt
3	gefüllter Kreisbogen
4	gefüllter Kreis
5	Ellipse
6	Ellipsen-Ausschnitt
7	gefüllter Ellipsen-Ausschnitt
8	Rechteck mit abgerundeten Ecken

9 gefülltes Rechteck mit gerundeten Ecken
10 justierter Text

Die erste Funktion davon, das gefüllte Rechteck, eignet sich hervorragend zur Erstellung von Balkendiagrammen. Dabei lassen sich beliebig breite Balken mit beliebiger Füllung darstellen, was für solche Diagramme enorme Möglichkeiten der Gestaltung bietet. Ein solches Rechteck läßt sich durch folgendes Programm darstellen:

```
10   rem *** gefülltes Rechteck ***
20   color 1,2,2,9,2          : rem Füllatribute
30   poke contrl,11           : rem Rechteck füllen
40   poke contrl+2,2          : rem Parameteranzahl
50   poke contrl+6,0
60   poke contrl+10,1         : rem Funktions-ID
70   poke ptsin,50            : rem X-Start-Koordinate
80   poke ptsin+2,50          : rem Y-Start-Koordinate
90   poke ptsin+4,100         : rem X-Ziel-Koordinate
100  poke ptsin+6,200         : rem Y-Ziel-Koordinate
110  vdisys 0                 : rem und Ausführung
```

Hierbei sind die Start-Koordinaten die des oberen, linken Punktes des Rechtecks, Ziel ist unten rechts. Die Bedeutung der Füllattribute wird später erklärt.

Die nächste Funktion, der Kreisausschnitt, entspricht dem BASIC-Befehl CIRCLE X,Y,R,A,E und bringt einen ungefüllten Kreisausschnitt auf den Bildschirm. Hierfür muß zusätzlich zu dem Beispiel des Kreises der Anfangs- und Endwinkel des Ausschnittes definiert werden. Diese Winkel werden in 1/10 Grad angegebenen, wobei der Nullpunkt rechts neben dem Mittelpunkt liegt. Ein Viertelkreis im ersten Quadranten, also oben rechts vom Mittelpunkt, liegt somit zwischen 0 und 900 (90 Grad). Dasselbe gilt auch für den CIRCLE-Befehl. Ein Beispiel:

```
10   rem *** Kreisausschnitt ***
20   color 1,0,2,1,1         : rem Linien-Attribute
30   poke contrl,11          : rem Kreisausschnitt zeichnen
40   poke contrl+2,4         : rem Parameteranzahl
50   poke contrl+6,2
60   poke contrl+10,2        : rem Funktions-ID
70   poke ptsin,150          : rem X-Mittelpunkt
80   poke ptsin+2,50         : rem Y-Mittelpunkt
90   poke ptsin+4,0          : rem Dummy
100  poke ptsin+6,0
110  poke ptsin+8,0
120  poke ptsin+10,0
130  poke ptsin+12,30        : rem Radius
140  poke intin,0            : rem Anfangswinkel
150  poke intin+2,900        : rem Endwinkel
160  vdisys 0                : rem und Ausführung
```

Dasselbe Programm kann auch einen ausgefüllten Kreisausschnitt malen. Dazu braucht nur die Funktions-ID 3 eingesetzt und die entsprechenden Füllattribute in Zeile 20 korrigiert zu werden. Für einen ganzen Kreis muß hier natürlich nicht 0-360 Grad definiert werden, sondern es wird die Funktion 4 verwendet. Diese hatten wir in unserem ersten Beispiel bereits vorgestellt.

Nun zu den verbogenen Kreisen, den Ellipsen. Hierfür stehen uns die Funktionen 5, 6 und 7 zur Verfügung. Nehmen wir als Beispiel hierfür die Funktion 6, die einen Ellipsen-Ausschnitt zeichnet, dessen Enden mit dem Mittelpunkt verbunden werden.

```
10   rem *** Ellipsen - Ausschnitt ***
20   color 1,0,1,1,1         : rem Linienattribute
30   poke contrl,11          : rem Ellipsenstück zeichnen
40   poke contrl+2,2         : rem Parameteranzahl
50   poke contrl+6,2
60   poke contrl+10,6        : rem Funktions-ID
70   poke ptsin,200          : rem X-Mitten-Koordinate
80   poke ptsin+2,100        : rem Y-Mitten-Koordinate
90   poke ptsin+4,100        : rem X-Radius
```

```
100  poke ptsin+6,50        : rem Y-Radius
110  poke intin,3200        : rem Startwinkel
120  poke intin+2,1200      : rem Endwinkel
130  vdisys 0               : rem und Ausführung
```

Für die Funktion 7 ändert sich lediglich die Funktions-ID und die Füllattribute, die ja mit dem COLOR-Befehl eingestellt werden können. Die Nummer 5 hingegen benötigt keine Anfangs- und Endwinkel, da eine vollständige Ellipse gezeichnet werden soll. Die Zeilen 110 und 120 fallen somit ersatzlos weg. In Zeile 50 wird eine 0 eingesetzt und die Funktions-ID wird durch die 5 ersetzt.

Als nächstes kommt eine Form an die Reihe, deren Zeichnung für ein BASIC-Programm ein großes Problem darstellt (es fehlt im Befehlssatz des BASIC-Interpreters die entsprechende Anweisung). Gemeint ist ein Rechteck mit abgerundeten Ecken. Hierfür stehen uns die ID-Nummern 8 und 9 zur Verfügung, deren Programmierung identisch ist. Der Unterschied zwischen beiden ist nur, daß das Rechteck bei Nummer 9 gefüllt ist und bei 8 nicht. Zu ändern ist dann lediglich die ID und die Füll-/Linienattribute des COLOR-Befehls.

```
10   rem *** Abgerundetes Rechteck ***
20   color 1,2,2,3,4        : rem Füllattribute
30   poke contrl,11         : rem Rechteck zeichnen
40   poke contrl+2,4        : rem Parameteranzahl
50   poke contrl+6,0
60   poke contrl+10,9       : rem Funktions-ID
70   poke ptsin,50          : rem X-Start-Koordinate
80   poke ptsin+2,50        : rem Y-Start-Koordinate
90   poke ptsin+4,200       : rem X-Ziel-Koordinate
100  poke ptsin+6,200       : rem Y-Ziel-Koordinate
110  vdisys 0               : rem und Ausführung
```

Das waren bisher nur die grafischen Grundfunktionen, die das GEM-VDI mit links beherrscht. Aber zur sinnvollen Anwendung von Grafiken gehören auch Beschriftungen. Auch dafür ist das VDI geeignet, denn es beherrscht eine große Anzahl von Möglichkeiten der Textdarstellung. Damit wären wir schon beim nächsten Thema:

7.1.2 Grafische Textgestaltung

Der letzte im Bunde der 11er-Aufrufe des VDI ist die Nummer 10. Diese Funktion erlaubt es, auf vielfältige Weise formatierten Text auf den Bildschirm zu bringen - natürlich auch an eine beliebige Stelle. Als Parameter bekommt diese Funktion einiges mit auf den Weg. Zum einen wäre da natürlich die Position, ab welcher der Text geschrieben werden soll. Diese Koordinaten zeigen auf die linke obere Ecke des ersten Buchstabens im Text. Weiterhin wird die gewünschte Gesamtlänge der Textausgabe übergeben. Ist diese länger als der Text, so wird dies durch gleichmäßiges Einfügen von Zwischenräumen korrigiert. Diese Ausdehnung kann beliebig ein- und ausgeschaltet werden, so daß man wählen kann, ob sich die Zwischenräume oder die Buchstaben selbst dehnen sollen.

Schließlich wird noch der Text selbst übertragen. Dieser muß dabei Zeichen für Zeichen in das INTIN-Array eingesetzt werden, was die ganze Sache etwas schwieriger macht als die obigen Zeichnungen. Doch das ist alles halb so wild. Hier ein Musterprogramm dafür:

```
10   rem *** Grafiktext - VDISYS-Demo ***
20   text$ = "Probetext"
30   poke contrl,11              : rem Befehlscode
40   poke contrl+2,3             : rem Anzahl der Parameter
50   poke contrl+6,len(text$)+2
60   poke contrl+10,10           : rem Funktionskennung
70   poke intin,1                : rem Wortdehnung (0=aus)
80   poke intin+2,1              : rem Zeichendehnung (0=aus)
90   poke ptsin  ,50             : rem X-Koordinate
```

```
100  poke ptsin+2,100           : rem Y-Koordinate
110  poke ptsin+4,50            : rem X-Textlänge
120  for i=1 to len(text$)      : rem ASCII-Zeichen
130  poke intin+2+i*2,asc(mid$(text$,i,1)) : rem  setzen
140  next i
150  vdisys 0                   : rem und Ausführung
```

Wo wir gerade bei Texten sind, sollten wir uns auch gleich einmal ansehen, welche Möglichkeiten der Textgestaltung das GEM bietet. Im Abschnitt 4.2.4 wurde bereits beschrieben, wie man eine der typischen Schriftarten des ATARI ST auswählt. Nun ist aber manchmal erwünscht Buchstaben oder Zahlen in unterschiedlichen Größen und Richtungen zu schreiben. Und auch hier ist das GEM der Helfer in der Not. Sehen wir uns dafür ein Programm an, das Riesenbuchstaben mitten auf den Bildschirm zaubert (deren Größe ist dabei noch nicht maximal...):

```
10   rem *** Zeichengröße ändern ***
20   poke contrl,12                 : rem Befehlscode
30   poke contrl+2,1                : rem Parameteranzahl
40   poke contrl+6,0
50   poke ptsin,0                   : rem Dummy
60   poke ptsin+2,30                : rem Zeichenhöhe
70   vdisys 0                       : rem und Ausführung
80   text$= "Was für ein Text !"
90   poke contrl,8                  : rem Text
100  poke contrl+2,2
110  poke contrl+6,len(text$)
120  poke ptsin,100                 : rem X-Koordinate
130  poke ptsin+2,100               : rem Y-Koordinate
140  for i=1 to
150  poke intin+2*i-2,asc(mid$(text$,i,1))
160  next i
170  vdisys 0                       : rem Text schreiben
180  poke contrl,12 : poke contrl+2,1
190  poke contrl+6,0 : poke ptsin+2,13
200  vdisys 0                       : rem und Normalisieren
```

Hier haben wir gleich zwei neue VDI-Kommandos vor uns, 8 und 12. Der Befehl 12 setzt die Zeichenhöhe für die auszugebenden Zeichen fest. Diese Höhe wird in PTSIN(1) übergeben und ist der einzige Parameter der Funktion. Die 0 in PTSIN(0) ist ohne Funktion, muß aber sicherheitshalber mit übertragen werden.

Der darauffolgende Programmteil ruft die Funktion 8 auf, die einen Text an den angegebenen X/Y-Koordinaten auf den Bildschirm schreibt. Hier wäre auch ein einfacher PRINT-Befehl möglich, der aber nur das OUTPUT-Fenster beschreibt.

Schließlich wird nach Ausgabe des Textes wieder das Standard-Format eingestellt. Dies muß unbedingt erfolgen, da der Bildschirmeditor mit dem riesigen Text wenig anzufangen weiß, so daß weitere Eingaben im Kommando- oder OUTPUT-Fenster nur noch Müll produzieren würden.

Mit dieser Funktion lassen sich schon einige interessante Effekte erzielen. Doch was machen wir, wenn in einer Zeichnung eine senkrechte Linie zu beschriften ist? Nun, auch hier hilft uns das VDI. Mit der Funktion 13 läßt sich die Grundlinie, auf der Texte dargestellt werden, in 90 Grad-Schritten einstellen. Dabei ist 0 wieder die Normaleinstellung. Der Winkel 900 läßt den Text von unten nach oben schreiben, 1800 stellt ihn auf den Kopf und 2700 bewirkt, daß er von oben nach unten lesbar wird. Leider unterstützt der Monochrome-Monitor des ATARI ST keine Zwischenwinkel, das ist Plottern vorbehalten.

Und so kann so etwas aussehen:

```
10  rem *** Zeichenrichtung ändern ***
20  poke contrl,13              : rem Befehlscode
30  poke contrl+2,0             : rem Parameteranzahl
40  poke contrl+6,1
50  poke intin,900              : rem 90 Grad
70  vdisys 0                    : rem und Ausführung
80  text$= "Senkrechter Text !"
90  poke contrl,8               : rem Text
```

```
100 poke contrl+2,2
110 poke contrl+6,len(text$)
120 poke ptsin,100
130 poke ptsin,300
140 for i=1 to len(text$)           : rem Text übertragen
150 poke intin+2*i-2,asc(mid$(text$,i,1))
160 next i
170 vdisys 0                        : rem Text schreiben
180 poke contrl,13 : poke contrl+2,0
190 poke contrl+6,1 : poke intin,0
200 vdisys 0                        : rem und Normalisieren
```

Auch hier wurde die Funktion 8, Textausgabe, mitverwendet. Der normale PRINT-Befehl tut es hier nicht mehr. Er schreibt nur halbe, liegende Buchstaben, was wohl nicht gerade brauchbar ist. Und es wurde, wie auch im vorherigen Beispiel, sofort wieder eine normalisierende Funktion aufgerufen - aus ähnlichen Gründen wie eben.

Soviel zur Textausgabe. Doch Kreise und Texte machen noch keine Grafik aus. Außerdem müssen Linien gezeichnet werden können - am besten in verschiedenen Linienarten. Auch das Zeichnen von Zeigern und Pfeilen wird in einer technischen Zeichnung dringend benötigt. Selbst das ist für das VDI kein Problem.

7.1.3 Linien- und Zeigerformen

Eine Linie in das Output-Fenster zu zeichnen ist die Aufgabe des LINEF-Kommandos. Angegeben werden hier Start- und Endposition der Linie. Die entsprechende Funktion des VDI ist die mit der Nummer 6, welche den Namen Polyline trägt. Polyline ist die Bezeichnung für mehrere zusammenhängende Linien. Der Funktion können nämlich mehrere Koordinaten-Punkte übergeben werden, die sie dann der Reihe nach verbindet. So können einfache Zeichnungen, wie Rahmen, einfach gezeichnet werden, da dabei nur ein Aufruf nötig ist. Hier wieder ein Beispielprogramm:

```
10   rem *** Mehrfach - Linie ***
30   poke contrl,6          : rem Polyline - Code
40   poke contrl+2,4        : rem Punkte-Anzahl
50   poke contrl+6,0
60   poke ptsin,50          : rem X 1
70   poke ptsin+2,50        : rem Y 1
80   poke ptsin+4,150       : rem X 2
90   poke ptsin+6,100       : rem Y 2
100  poke ptsin+8,160       : rem X 3
110  poke ptsin+10,250      : rem Y 3
120  poke ptsin+12,50       : rem X 4
130  poke ptsin+14,50       : rem Y 4
140  vdisys 0               : rem und Ausführung
```

Dieser Aufruf zeichnet ein Dreieck auf den Bildschirm. Hierzu war es natürlich nötig, den Start- bzw. Endpunkt zweimal anzugeben. In CONTRL(1) wird die Anzahl der übergebenen Koordinaten eingetragen. Für diese Funktion gibt es eine eng verwandte, deren Aufruf mit dem Befehlscode 9 geschieht. Dieser Aufruf zeichnet nicht nur den Polygon, sondern füllt ihn auch gleich aus. Zum Ausprobieren dieser Funktion muß nur noch zusätzlich ein COLOR-Befehl erfolgen, der die Füllattribute bestimmt.

Bleiben wir aber bei den Linien. Es wäre schön, wenn wir sie gepunktet oder gestrichelt darstellen könnten. Also wollen wir dies dem VDI klar machen. Hierzu dient die Funktion 15, mit der zwischen 7 verschiedenen Linienarten gewählt werden kann. Dieser Stil, der in INTIN(0) übergeben wird, bedeutet jeweils:

```
Stil                  Linie
 1    --------------------------------   durchgezogen
 2    ------------    ------------       unterbrochen
 3    ---      ---      ---      ---     gestrichelt
 4    -------   ---   -------   ---      Strich-Punkt
 5    --------        --------           lang gestrichelt
 6    ----   -- --    ----    -- --      Strich-Punkt-Punkt
 7                                       selbst definierbar
```

Stil 7 läßt sich beliebig selbst definieren. Die Definition besteht dabei aus einem 16 Bit-Wort, dessen Bitmuster die Linie bestimmt. Die durchgezogene Linie besitzt daher die Definition 1111111111111111 = $FFFF = 65535. Diese Zahl wird der Funktion 113 übergeben und bestimmt die Form jeder Linie, die danach mit der Stilnummer 7 gezeichnet wird. Hier ein Programm, das die beiden Funktionen Linie definieren und Linienart auswählen demonstriert:

```
10   rem *** Set Line Style ***
20   clearw 2              : rem OUTPUT-Fenster löschen
30   poke contrl,15        : rem Linienart wählen
40   poke contrl+2,0       : rem Parameteranzahl
50   poke contrl+6,1
60   poke intin,7          : rem Linienart User
70   vdisys 0              : rem und Ausführung
80   poke contrl,113       : rem Linie definieren
90   poke contrl+2,0       : rem Parameteranzahl
100  poke contrl+6,1
110  poke intin,65365      : rem Linienmuster
120  vdisys 0              : rem und Ausführung
200  linef 0,20,200,20     : rem Probelinie
```

Die Linien lassen sich auch in variabler Dicke darstellen. Dabei ist aber zu beachten, daß sich dickere Linien nur im Stil 1, also durchgezogen, darstellen lassen. Ist ein anderer Stil gewählt, so wird das einfach ignoriert. Die Strichdicke ist zwischen 1 und 15 einstellbar, indem man die Funktion 16 aufruft. Ein Beispiel:

```
10   rem *** Linien-Breite setzen ***
20   clearw 2              : rem OUTPUT-Fenster löschen
30   poke contrl,16        : rem Linienbreite setzen
40   poke contrl+2,2       : rem Parameteranzahl
50   poke contrl+6,0
60   poke ptsin,15         : rem Linienbreite
70   vdisys 0              : rem und Ausführung
100  linef 0,30,250,100    : rem Probelinie
```

Von diesen Linien lassen sich jetzt auch noch die Endungen definieren. Hierfür stehen 3 Gestaltungsmöglichkeiten zur Auswahl:

0	normal kantig
1	Pfeil
2	abgerundet

Eingestellt werden sie mit der Funktion 108, und zwar jeweils für Anfang und Ende der Linien getrennt. Um einen Pfeil von der Position X1/Y1 zur Position X2/Y2 zeigen zu lassen und dessen Anfang abzurunden, programmieren wir das folgendermaßen:

```
10  rem *** Pfeil zeichnen ***
20  poke contrl,108         : rem Endungen definieren
30  poke contrl+2,0
40  poke contrl+6,2
50  poke  intin,2           : rem Anfang abgerundet
60  poke intin+2,1          : rem Ende pfeilförmig
70  vdisys 0                : rem setzen
```

Danach führen wir das Zeichnen selbst wie im vorigen Beispiel aus. Die Abrundung eines Linienendes wirkt sich nur bei dicken Linien aus. Pfeile dagegen sind in jeder Strichbreite darstellbar.

7.1.4 Ausfüllen beliebiger Flächen

Hat man nach und nach eine Form erstellt und gezeichnet, so ist es schön, sie auszufüllen. Für die Grafikoperationen des BASIC im OUTPUT-Fenster ist das möglich mit dem FILL X,Y-Befehl, der an der angegebenen Stelle den Füllvorgang beginnt und alles innerhalb einer Rahmenlinie ausfüllt. Das Füllmuster und dessen Farbe wird vorher mit dem COLOR-Befehl eingestellt. Dieser Befehl erwartet 5 Parameter, deren Bedeutung folgende ist:

Parameter von COLOR A, B, C, D, E

A	Schriftfarbe von Texten
B	Füllfarbe für den FILL-Befehl
C	Linienfarbe für Zeichnungen
D	Füllmuster-Index
E	Füllmuster-Stil

Index und Stil stellen das Aussehen der Füllmuster ein. Dabei bedeutet der Stil in welcher Art die Füllung vorgenommen werden soll.

Stil	Füllart
0	Fläche wird nicht gefüllt
1	Fläche wird vollständig gefüllt
2	Füllung punktiert
3	Schraffur
4	Ausfüllung mit selbstdefiniertem Muster

Der Index spielt nur bei Stil 2 und 3 eine Rolle und gibt das Muster der Füllung an.

7.1.5 Erstellung eigener Füllmuster

Der Stil 4 ist nun sehr interessant. Ist es hier doch möglich, eine Fläche mit einem Muster nach unserem eigenen Geschmack zu füllen. Dabei können Firmensymbole für einen Briefkopf gewählt werden, oder auch der Bildschirm voller Herzchen gemalt werden. Der Phantasie sind dabei nur die Grenzen 16x16 gesetzt, die die Größe des Musters selbst begrenzen. Weil's so schön war, wollen wir als Beispiel unser, von der Mausform-Einstellung her bekanntes, Männlein als Füllmuster verwenden. Das Programm hierfür ist wieder so gestaltet, daß Sie sehr einfach Ihr eigenes Muster einsetzen können.

```
10   rem *** Füllmuster definieren ***
20   poke contrl,112               : rem Befehlscode
30   poke contrl+6,16              : rem Anzahl der Parameter
80   for i=0 to 15 : read x$       : rem Füllmuster
82   x=0 : for j=1 to 16           : rem Umwandlung
84   x= x-(mid$(x$,j,1)<>" ") * 2^(16-j)
86   next j
90   poke intin+i*2,x              : rem Muster setzen
100  next i
110  vdisys 0                      : rem und Ausführung
112  color 1,1,1,1,4               : rem Muster auswählen
114  pcircle 80,80,70              : rem und Demonstration
120  end
130  rem +++ Musterdaten +++
140  data "                "
150  data "      ***       "
160  data " **  * * *      "
170  data "  **  ***       "
180  data "   **  *        "
190  data "    ********    "
200  data "     ***** **   "
210  data "     *****  *   "
220  data "     ***** **   "
230  data "     *****      "
240  data "     ** **      "
250  data "     ** **      "
```

```
260 data "    ** **   "
270 data "    ** **   "
280 data "  **** ****  "
290 data "            "
```

Um nun dieses oder auch eines der vordefinierten Muster irgendwo auf den Bildschirm zeichnen zu können, ist wieder ein VDI-Aufruf nötig. Diesmal ist es die Funktion 103, die auch vom FILL-Befehl des BASIC aufgerufen wird. Die Handhabung ist also genauso wie bei FILL, was die Attribute-Einstellung und die Methodik des Füllvorgangs angeht. Ein solcher GEM-VDI-Aufruf sieht etwa so aus:

```
10  rem *** Füll -Demo ***
15  color 1,1,1,3,2       : rem Füll-Atribute
20  poke contrl   ,103    : rem Füllbefehl
30  poke contrl+2,2       : rem Parameteranzahl
40  poke contrl+6,1
50  poke intin    ,1      : rem Füllfarbe
60  poke ptsin    ,100    : rem X-Koordinate
70  poke ptsin+2  ,200    : rem Y-Koordinate
80  vdisys 0              : rem und Ausführung
```

Sie können damit übrigens auch die Menü-Leiste des GEM punktieren oder stricheln, wenn Sie diese persönliche Note bevorzugen.

7.1.6 Markierungen im Bild setzen

In einigen Programmen ist es erwünscht, verschiedene Markierungen auf den Bildschirm zu setzen, etwa um eine bestimmte Position anzuzeigen. Nun kann man natürlich mit dem CIRCLE-Befehl kleine Kreise malen oder mit einem geeigneten VDI-Graphikbefehl arbeiten. Dies ist aber nicht notwendig, da das VDI Markierungsbefehle besitzt. Die Form dieser Markierungen

läßt sich wieder aus einer ganzen Reihe von Vorgaben auswählen. Diese voreingestellten Typen sind:

Typ	Form
1	Punkt
2	Plus-Zeichen, etwa +
3	Stern, etwa *
4	Quadrat
5	Kreuz, etwa X
6	Diamant

Alle größeren Werte ergeben ebenfalls einen Stern. Bis auf den Punkt können diese Symbole auch noch vergrößert werden, so daß sie genau die gewünschte Größe ergeben. Mit dem entsprechend eingestellten Quadrat läßt sich dann z.B. ein Buchstabe umranden.

Nun ist die Auswahl des Typs, die Einstellung der Größe, die Wahl der Markierungsfarbe und das Setzen der Symbole jeweils eine eigene Funktion. Das erhöht den Aufwand für deren Verwendung, doch ist ihr Einsatz durch die große Flexibilität der Funktion dennoch oft von Vorteil. Viele Programme verwenden solche Markierungen auch zum Ankreuzen von ausgewählten Menüpunkten in Dialogboxen.

Lassen Sie uns deshalb ein Beispiel betrachten, in dem alle Einstellungen gemacht und dann einige Markierungen gesetzt werden:

```
10   rem *** Poly-Marker ***
20   poke contrl,18           : rem Markierungs-typ
30   poke contrl+2,0          : rem Parameter-Anzahl
40   poke contrl+6,1
50   poke intin,5             : rem Typ
60   vdisys 0                 : rem und setzen
70   poke contrl,19           : rem Größe
80   poke contrl+2,1
90   poke contrl+6,0
100  poke ptsin,0             : rem Dummy
```

```
110 poke ptsin+2,30        : rem Markierungshöhe
120 vdisys 0               : rem setzen
130 poke contrl,20         : rem Farbe setzen
140 poke contrl+2,0
150 poke contrl+4,1
160 poke intin,2           : rem Farbnummer
170 vdisys 0               : rem setzen

180 poke contrl,7          : rem Polymarker
190 poke contrl+2,2        : rem Punkte-Anzahl
200 poke contrl+6,0
210 poke ptsin,50          : rem X 1
220 poke ptsin+2,50        : rem Y 1
230 poke ptsin+4,150       : rem X 2
240 poke ptsin+6,100       : rem Y 2
250 vdisys 0               : rem und Zeichnen
```

Der Aufruf der Funktion 7, Markierung setzen, zeichnet gleich alle Symbole, deren Koordinaten übergeben wurden, gleichzeitig. Diese Koordinaten Xn/Yn können wieder irgendwo auf dem Bildschirm liegen. Alle weiteren Aufrufe der Funktion benutzen allerdings die eingestellten Attribute weiter.

Wenn man nun mit dieser Funktion und dem Typ 1 einige Punkte auf den Bildschirm gebracht hat, kommt es oft vor, daß man feststellen muß, ob sich an einer Stelle des Bildschirmes schon oder noch ein Punkt befindet. Auch das kann das VDI.

7.1.7 Bildpunkte austesten

Die Funktion 105 des VDI bietet die Möglichkeit, die Farbe eines beliebigen Punktes auf dem Bildschirm festzustellen. Ist diese Farbe die des Hintergrundes, so wird der Zustand "nicht gesetzt" zusätzlich gemeldet. Diese Informationen werden nach dem Aufruf der Funktion in INTOUT(0) und INTOUT(1) angeboten. Man kann mit ihnen einiges anfangen, wenn man z.B. in einem Spiel feststellen will, ob ein Objekt eine Begrenzungslinie berührt. Oder man kann sie verwenden, um Bild-Teile zu ko-

pieren. Ein Programm für die Abfrage eines Bildschirmpunktes sieht etwa so aus:

```
10   rem *** Bildpunkt austesten ***
20   poke contrl,105          : rem Befehlscode
30   poke contrl+2,2          : rem Parameteranzahl
40   poke ptsin,x             : rem X-Koordinate des Punktes
50   poke ptsin+2,y           : rem Y-Koordinate
70   vdisys 0                 : rem und Testen
80   gesetzt = peek(intout) : rem 1 = gesetzt / 0 = nicht
90   farbe = peek(intout+2) : rem 0/1 bei Monochrome
```

7.1.8 Farben mischen

Mit dem Kontroll-Paneel kann man die 16 Farben einstellen, die der ATARI ST auf einem Farbmonitor gleichzeitig darstellen kann. Diese Farben werden in einer Zeichnung verwendet, indem man sie mit der laufenden Nummer 0 bis 15 mittels des COLOR-Befehls auswählt. Nun ist es aber manchmal erforderlich, diese Farben in einem laufenden Programm zu variieren. Dafür fällt die Verwendung des Kontroll-Paneels natürlich aus. BASIC bietet für die Manipulation der einzelnen Farbregister keinen Befehl an. Dies muß also wieder über das VDI geschehen.

Die Funktion 14 des VDI erlaubt nun die exakte Einstellung eines beliebigen Farbregisters. Dies wird durch Mischung der 3 additiven Farbanteile der Farbe erledigt, wobei man die Intensität dieser Grundfarben einzeln einstellt. Die Abstufung reicht von 0 bis 1000, wodurch wirklich jede unmögliche Farbe möglich wird.

Die drei Intensitätswerte werden der Funktion gleichzeitig übergeben und durch Aufruf des VDI sofort eingestellt. Bereits gemalte Muster in der gewählten Farbe nehmen dann sofort auch den neuen Farbton an, da auch mit Tricks dem ST nicht mehr als 16 Farben zu entlocken sind!

Hier ein Programmbeispiel zur Einstellung der Farbnummer 2 auf einen Braunton:

```
10  rem *** Farbe einstellen ***
20  poke contrl,14       : rem Funktionsnummer
30  poke contrl+2,0
40  poke contrl+6,4
50  poke intin,2         : rem Farbnummer
60  poke intin+2,600     : rem Rot - Anteil
70  poke intin+4,400     : rem Grün-Anteil
80  poke intin+6,200     : rem Blau-Anteil
90  vdisys 0             : rem und einstellen
```

Um mit diesen Rot-, Grün- und Blauanteilen die gewünschte Farbe genau einzustellen, müssen Sie wohl ein wenig herumprobieren. Doch auch dies ist interessant, da man die große Farbenvielfalt des ATARI ST dabei entdeckt.

7.2 Geräusche und Musik

Der ATARI ST hat zur Erzeugung von Tönen und Geräuschen drei Tongeneratoren eingebaut, die sich beliebig mischen lassen. Zusätzlich läßt sich jeder dieser Tonkanäle als Rauschgenerator verwenden, indem man die entsprechende Programmierung des Sound-Chips vornimmt. Und schließlich hat dieser Chip noch einen Hüllkurvengenerator eingebaut, durch den sich recht einfach interessante Geräuscheffekte erzielen lassen.

Die Einstellungen dieser Tonvariationen lassen sich durch Kombination von SOUND- und WAVE-Befehlen erzielen. Diese Befehle bieten eine Fülle von Möglichkeiten ein Geräusch nach eigenen Vorstellungen recht realistisch herzustellen.

Die einfachste Art, einen Ton zu erzeugen, ist die, eine Taste zu drücken. Das hierbei ertönende Plop ist ebenfalls ein Ton, den der Sound-Chip herstellt. Dadurch erklärt sich, warum ein Tastendruck einen eingestellten Ton abbricht. Das Betriebs-

system programmiert nämlich, wenn ein Plop erzeugt werden soll, den Sound-Chip neu. Dadurch werden zumindest einige der vom Programmierer eingestellten Tonvariationen zurückgesetzt.

Der Tastaturklick und die Glocke, die bei einer Fehlermeldung ertönt (wenn dies im Kontrollfeld eingestellt ist), werden im Interrupt erzeugt. Das Betriebssystem braucht dafür nur einen Zeiger auf eine Tabelle zu stellen, die die Programmdaten für den Sound-Chip enthält. Das fast ständig ablaufende Interruptprogramm testet bei jedem Durchlauf diesen Zeiger, ob er auf 0 steht. Enthält er einen Wert, so wird dieser als Zeiger interpretiert. Die Interruptroutine nimmt nun die Werte aus der angezeigten Tabelle und programmiert mit ihnen den zu erzeugenden Ton. Stößt sie auf ein $FF in der Tabelle, so beendet sie ihre Arbeit und der normale Betrieb geht weiter.

Um dies auszuprobieren, lassen wir doch einmal einen dieser voreingestellten Töne erklingen. Dazu stellen wir den Zeiger, der in der Adresse $A86 liegt, auf die Tabelle im Betriebssystem. Für ein Tastaturklick schreiben wir:

```
10 defdbl a
20 a = 3652
30 poke a,36642
```

Ein Glockenzeichen ertönt durch:

```
30 poke a,36612
```

Natürlich können wir uns eine solche Tabelle selbst erstellen, um ein Geräusch oder eine ganze Tonfolge durch ein einziges POKE ertönen zu lassen. Bevor wir dies ausprobieren, sollten wir uns erst einmal die Programmierung des Sound-Chips näher ansehen.

Die Parameterübergabe an den Sound-Chip erfolgt im I/O-Bereich des Speichers. Der Haken dabei ist wieder einmal der, daß ein Zugriff zu diesem Bereich privilegiert ist und somit bei

einem einfachen POKE zum Absturz führt. Doch dafür haben wir ja bereits ein Patentrezept kennengelernt, nämlich die Umgehung dieses Problems durch ein kleines Maschinenprogramm, welches durch vorübergehende Umschaltung des Systems in den Supervisor-Modus den Zugriff erlaubt.

Ein solches Programm haben wir in Kapitel 6.4.1 schon geschrieben. Mit einem CALL-Befehl kann nach dessen Durchlauf ein solcher privilegierter Zugriff vorgenommen werden. Wir laden also nun das Programm von der Diskette und erweitern es um folgende Zeilen:

```
200 read r,w                  : rem Register und Wert
205 if r=-1 then end          : rem Test auf Ende
210 call b (16746496,r*256)   : rem Register wählen
215 call b (16746498,w*256)   : rem Wert einschreiben
220 goto 200                  : rem weitermachen...
249 rem         A und B einschalten
250 data 7,252
259 rem         Kanal A Lautstärke
260 data 8,12
269 rem         Kanal B auf Hüllkurve, Frequenz
270 data 9,16,3,2
279 rem         Frequenz der Hüllkurve
280 data 13,10
299 rem         Ende !
300 data -1,0
```

Dieses Programm erzeugt ein Tongemisch von zwei Frequenzen, wovon eine noch variiert durch Aufschalten einer Hüllkurve. Die Daten ab Zeile 250 haben nun folgende Bedeutung:

Es handelt sich jeweils um ein Datenpaar: erst die Registernummer des Sound-Chips und dann der Wert, der in dieses Register geschrieben werden soll. Hierbei haben wir die Möglichkeit, durch Variation der Werte zu experimentieren, um den gewünschten Ton zu erzeugen. Die Register und die entsprechen-

den Inhalte haben unterschiedliche Bedeutungen, die wir nun betrachten wollen.

Register Nr.	Bedeutung
0 und 1	Periodendauer des Kanals A. Es werden insgesamt 12 Bit dieses Wortes benutzt
2 und 3	s.o., nur Kanal B
4 und 5	s.o., nur Kanal C
6	Periodendauer des Rauschgenerators. Es werden die unteren 5 Bit genutzt
7	Konfigurationsregister. Jedes Bit hat hier eine eigene Aufgabe: Bit 0: Kanal A an/aus (0=an, 1=aus) Bit 1: Kanal B an/aus (0=an, 1=aus) Bit 2: Kanal C an/aus (0=an, 1=aus) Bit 3: Kanal A mit Rauschen (0=ja, 1=nein) Bit 4: Kanal B mit Rauschen (0=ja, 1=nein) Bit 5: Kanal C mit Rauschen (0=ja, 1=nein) Bit 6: Port A Datenrichtung (0=in, 1=out) Bit 7: Port B Datenrichtung (0=in, 1=out)
8	Lautstärke Kanal A. Es gelten die unteren 4 Bits. Ist Bit 4 gesetzt, wird dem Kanal die Hüllkurve aufgeschaltet und die unteren 4 Bits ignoriert
9	s.o., nur Kanal B
10	s.o., nur Kanal C
11 und 12	Periodendauer der Hüllkurve (LO, HI). Alle 16 Bit werden genutzt.

13	Kurvenform der Hüllkurve (Bits 0 bis 3). Diese Kurvenformen sind schwer zu beschreiben, deshalb ausprobieren!
14	Port A Datenregister
15	Port B Datenregister

Die beiden Ports A und B sind frei programmierbare Datenregister, die im ATARI ST einige Aufgaben haben, die eigentlich nichts mit der Tonerzeugung zu tun haben. So ist Port B direkt an den Parallelport des Druckers angeschlossen und steuert diesen, während Port A die Selektierung der Diskettenstation, die Datenanforderung der seriellen Schnittstelle und den GPO-Anschluss des Monitorsteckers steuert. Die Manipulation dieser Register ist deshalb nicht ratsam. Bleiben wir aber bei der Tonerzeugung.

Wenn man nun so lange herumexperimentiert hat, daß man die Toneffekte nach Wunsch hinbekommt, so kann man sich die Sache enorm erleichtern. Wenn wir nämlich eine Tabelle für die Register und deren gewünschten Inhalt aufstellen und an einem definierten Speicherplatz ablegen, so können wir den oben erwähnten Zeiger der Sound-Interrupt-Routine auf diese Tabelle stellen. Der Ton oder auch eine ganze Tonfolge wird dann durchgespielt, während das BASIC-Programm weiterarbeiten kann. So kann z.B. eine Hintergrundmusik gespielt werden oder, wie man es aus Spielen kennt, ein Schuß- oder Treffergeräusch eingespielt werden, während das Spiel weitergeht.

Doch mit den Tönen und Geräuschen ist die Möglichkeit dieser Technik der Tonerzeugung noch nicht ausgeschöpft. So können im Laufe der Musik auch Pausen programmiert werden, die sich ebenfalls nur auf die Tonerzeugung beziehen und nicht auf unser BASIC-Programm. Des weiteren können steigende oder fallende Werte direkt vorprogrammiert werden, so daß sich z.B. eine Sirene mit wenig Aufwand programmieren läßt. Sehen wir uns dafür ein Beispielprogramm an, das ungefähr ein typisches Fallgeräusch mit folgendem Aufschlag produziert.

```
10   rem ** Tonsequenz erstellen **
20   def seg=0
30   defdbl b : b=3652          : rem Langwort einstellen
40   a$=space$(100)             : rem Platz schaffen
50   a=peek(varptr(a$)+2)       : rem Adresse feststellen
60   def seg=1
70   for i=0 to 100
80   read x                     : rem Werte übertragen
90   poke a+i-1,x
100  if x=255 then 120
110  next i
120  poke a+i,0                 : rem Abschluss-Null
130  def seg=0
140  poke b,a                   : rem und Ton starten!

150   rem -- Tondaten --
160   data 0,1,1,0,2,0,3,0
170   data 4,0,5,0,6,0,7,254
180   data 8,16,9,0,10,0,11,0
190   data 12,35,13,10

195   data 128,50
200   data 129,0,1,250

205   data 200,30

210   data 0,00,1,2,2,0,3,0
220   data 4,0,5,0,6,0,7,246
230   data 8,16,9,0,10,0,11,0
240   data 12,62,13,9
250   data 255
```

Der erste Block von Daten in den Zeilen 160 bis 190 erstellt einen Ton, dessen Lautstärke durch Modulation mit einer Dreieckskurve fällt. Das wird durch Einstellung der Lautstärke von Kanal A auf 16 in Register 8 bewirkt, was die Abhängigkeit von der Hüllkurve bewirkt. Die Hüllkurve selbst wird in Register 11, 12 und 13 eingestellt, was aus der Tabelle der Sound-Chip-Register zu entnehmen ist.

Nun zu den Zeilen 195 und 200. Hier werden zwei Sonderbefehle für den Sound-Interrupt verwendet. Ein solcher Sonderbefehl wird anstelle eines Registers angegeben und ist immer größer als 127. Der Sonderbefehl 128 bewirkt, daß der darauffolgende Wert zwischengespeichert wird (mehr nicht). Interessant ist dieser Wert allerdings in Verbindung mit dem nächsten Sonderbefehl. Die 129 läßt nun folgenden Ablauf starten:

Die nächste Zahl wird zunächst als Register interpretiert, um das es im folgenden gehen soll. Dieses Register wird zuerst selektiert, und dann wird der vorhin mit dem 128-Befehl abgespeicherte Wert hineingelegt. In unserem Beispiel wird die Periodendauer des Kanals A und damit seine Frequenz eingestellt. Es startet somit ein Ton, der in Abhängigkeit der Hüllkurve seine Lautstärke variiert.

Nun wird als nächstes ständig der auf die Registerkennzahl folgende Wert zu diesem Register addiert. Das führt dazu, daß die Tonhöhe kontinuierlich fällt, da ein steigender Wert eine fallende Frequenz zur Folge hat.

Die letzte Zahl, die zu dem Sonderbefehl 129 gehört, ist nun der Endwert, den das Register maximal bekommen soll. In unserem Beispiel bedeutet das den tiefsten Ton, den der Kanal A erreichen soll.

Wenn das Timing des obigen Beispiels genau eingestellt ist, so ist der Ton gerade dann an seiner tiefsten Frequenz angelangt, wenn auch die Hüllkurve ihr Maximum erreicht hat. Wenn man sich dabei einen fallenden Körper vorstellt, so muß dieser jetzt aufschlagen. Um es etwas spannender zu machen, wird aber erst eine kleine Pause eingelegt, bevor das geschieht. Eine solche Pause bewirkt der dritte Sonderbefehl. Dieser muß irgendein Wert über 129 sein. Die darauffolgende Zahl stellt die Pausenlänge ein. In dieser Zeit ändert sich nichts am vorhergehenden Zustand der Soundchip-Register und damit am Ton. Man kann damit ebenso wie mit dem WAVE-Befehl eine Verzögerung einstellen, bevor ein Ton erklingt.

Mit diesem Trick der Tonerzeugung über Interrupt sind jede Menge Möglichkeiten für die Realisierung einer geräuschvollen Umgebung eines Spieles offen. Man sollte dabei aber beachten, daß ein Tastendruck den Interrupt-Zeiger wieder umbiegt und dabei den aktuellen Ton abschaltet. Das gilt allerdings genauso für die Geräusche, die mit SOUND und WAVE eingestellt werden. Diese beiden Befehle wollen wir uns nun zum Abschluß noch ansehen.

Der SOUND-Befehl kann 4 oder 5 Parameter übernehmen. Diese Werte haben bei der Anweisung SOUND A, B, C, D, E folgende Bedeutung:

A	Kanal-Nummer (0-2)
B	Lautstärke (0-15)
C	Note des Tons (1-12)
D	Oktave des Tons (1-8)
E	Tondauer (0-255 in 1/50 Sekunden, kann entfallen)

Die Frequenzeinstellung ist also in Oktaven und Noten aufgeteilt, wodurch die Übertragung eines Musikstückes nach Noten sehr leicht wird.

Der WAVE-Befehl benötigt ebenfalls 4 bis 5 Parameter. Hier bedeutet WAVE A, B, C, D, E folgendes:

A	Konfigurieren. Hier wird das Register 7 des Soundchips eingestellt. Allerdings werden nur die unteren 6 Bits angenommen.
B	Hüllkurve ein-/ausschalten
C	Hüllkurvenform einstellen, Register 13
D	Periodendauer der Hüllkurve einstellen
E	Verzögerung des Tones (kann entfallen)

Mit diesen beiden BASIC-Befehlen können Sie nun auch einmal versuchen, das Fallgeräusch aus obigen Beispielprogramm herzustellen. Es ist erstaunlich, welche Vielzahl von Geräuschen man dem ATARI ST entlocken kann!

7.3 Fenster- und Menüprogrammierung

Was den ATARI ST so hervorhebt, ist sicher besonders die sehr einfache Bedienung, da man mit Menüs arbeitet und seinen Schreibtisch beliebig konfigurieren kann. Diese Technik können sich auch BASIC-Programmierer zunutze machen.

Die einfachste Methode, ein Menü herzustellen, ist die, mehrere Auswahlpunkte mit eine Kennziffer zu versehen und auf den Bildschirm zu schreiben. Der Benutzer wird dann aufgefordert, die Kennziffer seiner Wahl einzugeben. Das Programm kann dann mit dieser Ziffer den Programmteil abarbeiten lassen, der gewählt wurde. Das ganze sieht dann etwa so aus:

```
10   print "1) Daten eingeben"
20   print "2) Daten drucken"
30   print "3) Ende"
40   input "Bitte wählen ";w
60   on w goto 100,200,80
70   goto 40
80   end
100  rem ** Dateneingabe **
110  .....
200  rem ** Daten ausdrucken **
210  .....
```

Diese Art der Menüführung wird in sehr vielen Programmen angewendet. Der ATARI ST bietet jedoch die Möglichkeit, dies wesentlich komfortabler zu gestalten.

Da wären erst einmal die 10 Funktionstasten. Mit ihnen kann eine Auswahl erfolgen, die dem obigen Beispiel entspricht, jedoch komfortabler in der Handhabung ist. So muß die Auswahlziffer nicht mit der Returntaste abgeschlossen werden wenn man anstelle des INPUT-Befehls schreibt:

```
50 w=inp(2)-186
```

Da die F1-Taste den Wert 187 liefert, wird durch die Subtraktion daraus eine 1, mit der die ON...GOTO-Verzweigung ebenfalls erfolgen kann. Dies wäre also der erste Schritt der Vereinfachung der Bedienung eines Menüs. Diese Technik wird auch in vielen Programmen bereits verwendet, wie z.B. bei SM-TEXT, dem Textverarbeitungsprogramm, auf dem auch dieses Buch geschrieben wurde.

Doch gehen wir den nächsten Schritt zum perfekten Menü. Das Betriebssystem des ATARI ST bietet mit seinem GEM für die Menüführung etliche Möglichkeiten an. Für die Programmierung des GEM haben wir ja bereits den VDISYS-Befehl verwendet. Das VDI hilft uns hier aber wenig, da es nur für die Graphik als solches zuständig ist. Wir nehmen jetzt den geheimnisvollen GEMSYS-Befehl unter die Lupe.

GEMSYS aktiviert ebenso wie VDISYS das GEM im ATARI ST. Der Unterschied ist dabei der, daß VDISYS nur das VDI aufruft, wogegen GEMSYS für das AES zuständig ist. Das AES ist zuständig für die Fenster- und Menüverarbeitung im ST. Es ermöglicht also die luxuriöse Benutzerführung mit der Maus.

Ebenso wie der VDISYS-Befehl benötigt GEMSYS einige Parameter, um zu wissen, was es zu tun hat. Diese Parameter werden ebenfalls in Arrays übergeben, die die Namen CONTRL, INTIN, INTOUT, ADDRIN, ADDROUT tragen. AHA, denkt sich der erfahrene VDI-Programmierer, die ersten drei davon kenne ich ja schon. Irrtum! Hier gibt es nämlich ein großes Problem. CONTRL, INTIN und INTOUT sind hier nicht identisch mit den bekannten Arrays des VDI! Diese Datenfelder liegen im Speicher auch an einer anderen Adresse.

Die Frage ist nun, wie wir an diese Adressen herankommen. Die Systemvariablen des BASIC für CONTRL und die anderen können wir nicht benutzen, und ADDRIN bzw. ADDROUT kennt das BASIC überhaupt nicht. Was es jedoch noch kennt und was unserer Beachtung bisher entgangen ist, ist die Systemvariable GB. Dies ist, genau wie die anderen auch, ein Zeiger. Er zeigt jedoch nicht auf ein Datenfeld, sondern auf eine

Tabelle, in der wiederum Zeiger liegen. Und genau diese Zeiger brauchen wir.

Es handelt sich dabei um 32-Bit-Zeiger, deren Reihenfolge ihrer Bedeutung nach folgende ist:

PEEK(GB)	CONTRL
PEEK(GB+4)	GLOBAL
PEEK(GB+8)	INTIN
PEEK(GB+12)	INTOUT
PEEK(GB+16)	ADDRIN
PEEK(GB+20)	ADDROUT

Diese Zeiger werden dann für die Parameterübergabe an das AES bzw. für den GEMSYS-Befehl verwendet. Das GLOBAL-Array spielt hierbei nur eine untergeordnete Rolle. Es enthält Informationen über den Status des momentan bearbeiteten GEM-AES-Befehls. Hier die einzelnen Bedeutungen der Einträge im GLOBAL-Feld:

GLOBAL	GEM-Versionsnummer
GLOBAL+2	maximale Anzahl der gleichzeitig aktiven Programme
GLOBAL+4	Nummer des aktuellen Programms
GLOBAL+10	Zeiger auf eine Baumstruktur

Dieses Feld können wir im allgemeinen vergessen. Wichtiger sind die anderen Arrays, von denen das CONTRL-Feld die gleiche Aufteilung hat wie beim VDI. CONTRL(0) erhält wieder die Funktionsnummer, CONTRL(1) und CONTRL(2) die Anzahl der INTIN- und INTOUT-Einträge und CONTRL(3) und CONTRL(4) die Anzahl der Einträge des ADDRIN- und ADDROUT-Feldes in Langworten.

Beginnen wir die Anwendung des GEMSYS-Befehls mit einem sehr einfachen Aufruf. Es handelt sich bei dem folgenden Beispiel um die Funktion FORM_ERROR, die eine Fehlermeldung des TOS in einem kleinen Fenster mitten auf dem Bildschirm ausgibt. Die Fehlernummer wird dabei in INTIN(0) übergeben.

```
10   rem *** TOS-Error - Demo ***
20   defdbl b : b=gb
30   cn =peek(b)        : rem Zeiger auf CONTRL
40   ii =peek(b+8)      : rem Zeiger auf INTIN
50   poke cn,53         : rem Funktionsnummer
60   poke cn+2,1
70   poke cn+4,1
80   poke cn+6,0
90   poke cn+8,0
100  poke ii,22         : rem Fehlernummer
120  gemsys 53          : rem und Darstellung
```

Nun sehen wir uns dieses Beispiel einmal genauer an. Zuerst fällt auf, daß nicht direkt mit GB gearbeitet wird, sondern der Umweg über die Variable B genommen wird. Der Grund hierfür liegt darin, daß die Zeiger CN und II Langworte sein müssen. Ein PEEK(GB) gibt jedoch nur das HI-Wort des Zeigers CONTRL wieder, mit dem man nichts anfangen kann. Da aber die Variable B als doppelt genau definiert wurde, ergibt das PEEK(B) ein Langwort.

Als nächstes könnte man fragen, warum nicht als Variablennamen CONTRL und INTIN verwendet werden, was das Programm übersichtlicher machen würde. Leider sind ja gerade diese beiden Namen für die Systemvariablen reserviert, so daß sie zur andersgearteten Verwendung nicht zulässig sind.

Noch ein Unterschied zur VDI-Programmierung liegt hier vor. Der GEMSYS-Befehl wird direkt mit der Funktionsnummer geschrieben, der Wert nach GEMSYS ist also hier kein Dummy, sondern die Funktionsnummer selbst.

Die Darstellung einer Meldung TOS-ERROR ist nun leider nicht besonders brauchbar für ein BASIC-Programm. Hier wäre es interessanter, einen anderen Text darzustellen und vielleicht noch mehrere Auswahlpunkte zur Verfügung zu stellen. Dies ist auch durch die AES-Funktion FORM_ALERT möglich, die zwar eigentlich auch nur für Fehlermeldungen vorgesehen ist, dennoch für andere Anwendungen brauchbar ist. Dieser Funktion werden zwei Parameter übergeben. Der erste davon ist ein Zeiger auf einen Text, der die Mitteilung, die Aufschriften der (maximal 3) Auswahlpunkte und die Nummer des darzustellenden Symbols neben der Meldung.

Diesen Text kann man einfach in einer Textvariablen ablegen und der AES-Funktion die Adresse dieser Variablen mitteilen. Außerdem wird noch ein anderer Parameter übergeben, der bestimmt, welcher der Auswahl-"Knöpfe" auch mit der Return-Taste ausgewählt werden kann. Dieser wird dann stark umrandet dargestellt, wie wir das z.B. von der Programmauswahl nach LOAD oder SAVE her kennen. Dabei ist der OK-Knopf hervorgehoben, was heißt, daß ein Druck auf die Returntaste die Operation stattfinden lassen soll.

Hierbei ergibt sich jedoch ein Problem. Die Funktion gibt in INTOUT(0) die Nummer des ausgewählten Knopfes zurück. Bevor man diesen Wert jedoch mit PEEK(IO) abfragen kann, wird er durch einen weiteren Aufruf, den das BASIC vornimmt, wieder verändert. Somit ist es mir nicht gelungen, an die Nummer heranzukommen. Die einzige Möglichkeit, die uns hier bleibt, ist die, in INTIN(0) eine 0 zu übergeben. Dies bewirkt, daß keine der Auswahlmöglichkeiten durch die Returntaste gewählt werden kann. Dadurch kann diese Auswahl nur mit dem Mauszeiger erfolgen, dessen Position man bei der Rückkehr aus der AES-Funktion sofort feststellen und auswerten kann. Dabei kann man die Y-Koordinate des Mauszeigers vernachlässigen, da die Auswahlpunkte grundsätzlich nebeneinander dargestellt werden.

Der Funktionsaufruf mit der Erstellung des benötigten Textes wird nun in folgendem Beispielprogramm vorgestellt:

```
10   rem *** Alert - Demo ***
11   defdbl b,d : b=gb
12   cn=peek(b)          : rem Zeiger definieren
14   ii=peek(b+8)
20   io=peek(b+12)
25   d =peek(b+16)
30   a$ = "[1][Sie haben die|Wahl :............]"
31   a$=a$+"[Taste 1|Taste 2|Taste 3]"
40   a = peek(varptr(a$)+2)
50   poke cn,52          : rem Funktionsnummer
60   poke cn+2,1
70   poke cn+4,1
80   poke cn+6,1
90   poke cn+8,0
100  poke ii,0           : rem keine 'Return'-Auswahl
110  poke d,a            : rem Textadresse
120  gemsys 52           : rem Ausführung
125  t=peek(io)          : rem eigentlich Tastennummer...
```

Die 1 innerhalb der ersten eckigen Klammer im Text steht dabei nun für das Symbol, welches neben dem Text der Mitteilung stehen soll. Sie haben dabei die Auswahl zwischen Symbolen wie einem STOP-Schild, einem Fragezeichen oder einem Ausrufungszeichen.

Die zweite Klammer enthält dann den Text der Mitteilung. Die Zeilentrennung wird dabei durch den senkrechten Strich markiert. Die Länge des Textes sollte so gewählt werden, daß das Fenster mit den Auswahlknöpfen richtig proportioniert ist. Diese Knöpfe werden durch den Inhalt der dritten Klammer im Text definiert. Es können maximal 3 Auswahlmöglichkeiten dargestellt werden, welche höchstens 20 Zeichen lang sein dürfen. Die Trennung der einzelnen Einträge wird wieder durch den senkrechten Strich markiert. Die Zeile 125 des obigen Programms kann entfallen. Sie wäre nur sinnvoll, wenn uns das BASIC nicht den Wert von INTIN(0) verändern würde. Die Auswertung, welcher Knopf gewählt wurde, muß daher leider zusätzlich erfolgen.

Um nun die Position der Maus abzufragen, haben wir ja schon mehrere Möglichkeiten kennengelernt. Da wir aber gerade bei der Besprechung des GEM-AES sind, wollen wir diese Abfrage auch durch einen GEMSYS-Befehl durchführen.

Die Funktion dafür gibt dem aufrufenden Programm noch mehr als nur die Position des Mauszeigers zurück. Man erhält zusätzlich noch den Zustand der beiden Maustasten und den der Shift-, Control- und Alternate-Tasten. Der Aufruf benötigt keine Übergabeparameter.

```
10    rem *** Maus-Status ***
11    defdbl b : b=gb
12    cn=peek(b)
25    io=peek(b+12)
50    poke cn,79           : rem Funktionsnummer
60    poke cn+2,0
70    poke cn+4,5
80    poke cn+6,0
90    poke cn+8,0
100   gemsys 79            : rem Aufruf des AES
110   for i=2 to 8 step 2
120   x(i)=peek(io+i) : next i
130   print " Position  ";x(2);" , ";x(4)
140   print " Maustaste ";x(6)
150   print " Taste     ";x(8)
```

Der Umweg über die Variablen X wird hierbei gemacht, da ein PRINT-Befehl, der z.B. die Position der Maus direkt ausgeben sollte, das AES selbst aufrufen und damit die Parametertabellen verändern würde. Interessant dabei ist, daß nach dem Aufruf der Funktion 52 das INTOUT-Feld durch genau diese Funktion 79 zerstört wird. Deshalb kann die Auswertung der Mausposition ohne einen weiteren GEMSYS-Befehl erfolgen.

Die oben vorgestellten AES-Funktionen erstellen selbsttätig ein kleines Fenster auf dem Bildschirm, ohne daß man sich über die gewünschte Position oder Größe des Fensters den Kopf zerbrechen muß. Nun bietet das BASIC aber vier variable Fenster, deren Größe und Position mit Hilfe des Mauszeigers veränderbar ist. Um eines dieser Fenster aus einem Programm heraus zu verändern, wird ein weiterer AES-Aufruf benötigt. Es handelt sich dabei um die Funktion 105, die den Namen WIND_SET trägt. Mit ihr läßt sich ein Fenster nicht nur verschieben, sondern es wird auch möglich, den Randbereich des Fensters zu konfigurieren. Damit läßt sich z.B. verhindern, daß es mit der Maus gelöscht wird.

Hier ein Beispielprogramm, welches die Position und Größe des LIST-Fensters einstellt. Durch Einsetzen einer anderen Zahl in Zeile 40 kann auch eines der anderen Fenster verändert werden.

```
10   rem *** Fenster einstellen ***
20   defdbl b : b=gb
30   ii=peek(b+8)
40   h=2             : rem LIST-Fenster
50   poke ii,h       : rem wählen
60   poke ii+2,5     : rem Modus
70   poke ii+4,20    : rem X-Koordinate
80   poke ii+6,20    : rem Y-Koordinate
90   poke ii+8,350   : rem Breite des Fensters
100  poke ii+10,100  : rem Höhe
110  gemsys 105      : rem und setzen
```

Die Koordinaten geben die Position der linken, oberen Ecke des Fensters an. Der Modus 5, der in Zeile 60 übergeben wird, stellt die Funktion ein, die das AES ausführen soll. Dabei sind folgende Modi erlaubt:

Modus 1: in ii+4 wird eine Zahl erwartet, deren Bitkombination den Randbereich des Fensters definiert. Dabei bedeuten die Bits:

Bit 0:	Titelzeile des Fensters
Bit 1:	Löschfeld
Bit 2:	Öffnungsfeld
Bit 3:	Bewegungsfeld
Bit 4:	Informationszeile
Bit 5:	Größenveränderungs-Feld
Bit 6:	Pfeil nach oben
Bit 7:	Pfeil nach unten
Bit 8:	Vertikaler Schieber
Bit 9:	Pfeil nach links
Bit 10:	Pfeil nach rechts
Bit 11:	Horizontaler Schieber

Modus2: Der Name des Fensters wird geändert. Die Adresse des neuen Namens wird dabei in ii+4 bis ii+7 als Langwort erwartet. Der Text muß mit einer Null abgeschlossen sein.

Modus 3: Die Informationszeile des Fensters wird geändert. Die Bedingungen sind wie in Modus 2.

Modus 5: Das Fenster wird eingestellt. Dieser Modus wird in unserem Beispiel verwendet.

Modus 8 und 9: Die relative Position des horizontalen bzw. vertikalen Schiebers wird eingestellt.

Modus 10: Das aktuelle Fenster wird gewählt.

Modus 15 und 16: Die relative Größe des horizontalen bzw. vertikalen Schiebers wird eingestellt.

Mit diesen Funktionen läßt sich also ein beliebiges Fenster in jede gewünschte Form bringen. Dadurch kann ein BASIC-Programm wieder etwas flexibler gestaltet werden.

7.4 Textverarbeitung

Texte spielen in der Computeranwendung eine sehr große Rolle. Ein Computer kann ja auch schließlich mehr als einfach nur rechnen. Er kann auch Texte verwalten, verändern, speichern und drucken. Doch auch für andere Zwecke können Texte, in unserem Fall Textvariablen, angewendet werden. Man kann in ihnen auch andersgeartete Daten ablegen, wie z.B. Maschinenprogramme. Diese Methode haben wir ja bereits schon kennengelernt.

Um einen Speicherbereich des Computers in eine Textvariable zu legen, kann mit dem Variablenzeiger VARPTR() und POKE gearbeitet werden. VARPTR(A$) zeigt z.B. auf eine Tabelle, in der einige Informationen über die Variable A$ stehen. Der zweite Eintrag stellt dabei die Speicheradresse der Variablen selbst dar. Man erhält sie somit durch den Befehl A=PEEK(VARPTR(A$)+2). Um in diese Variable, die auch String genannt wird, eine Datei von der Diskette einzuladen, kann man folgendes Programm verwenden:

```
10   rem *** Datei in String ablegen ***
15   a$=space$(200)           : rem Platz reservieren
17   a=peek(varptr(a$)+2)     : rem Adresse feststellen
20   input "Filename ";fn$    : rem Dateinamen eingeben
30   bload fn$,a              : rem und einlesen
40   print :o=(peek(a) and 255)   : rem Offset errechnen
50   e=o+peek(a+4)            : rem Ende errechnen
60   for i=o+2 to e step 2
70   print i ,peek(a+i)       : rem Inhalt anzeigen
80   next i
```

Mit diesem Programm können kleine Diskettendateien (<200 Bytes) in A$ eingelesen werden. Die Zeilen 40 bis 80 sind hierzu nicht nötig, sie sind nur in Verbindung mit einem Applikationsprogramm (*.PRG) sinnvoll. Läd man ein solches ein, so wird Anfang und Ende des eigentlichen Programmes berechnet und dieses als Dezimalworte ausgegeben. Dieses Programm habe ich dazu verwendet, die kleinen Maschinenprogramme, die mit einem Assembler erstellt wurden, in DATA-Zeilen umzuwandeln. Dazu braucht man nur die ausgegebenen Zahlen in DATA-Zeilen einzutippen und mit der READ-POKE-Schleife nachher in den String wieder einzulesen.

Man kann diese Technik der Abspeicherung einer Variablen auch dafür verwenden, ein größeres Textfeld auf Diskette abzulegen. Man kann dies zwar auch mit OPEN und PRINT# machen, aber das dauert doch wesentlich länger als BLOAD bzw. BSAVE.

Doch nun wollen wir die eigentliche Aufgabe von Textvariablen und deren Verwendung nicht vergessen. Eine davon ist es, Einträge zu speichern und ggf. zu sortieren. Ein solches Sortierprogramm ist mit dem BASIC des ATARI ST recht einfach, da es so schöne Befehle wie SWAP kennt. Hier ein Beispiel, wobei der sogenannte Bubble-Sort-Algorithmus angewandt wird:

```
10   rem *** Sortieren von Texten ***
20   dim w$ (10)
30   for x=1 to 10
40   print x;". Wort :";:input w$ (x)
50   next x
60   print "Sortiervorgang läuft"
70   for i=1 to 10
80   for j=i to 10
90   if w$ (j)<w$ (i) then swap w$(i),w$(j)
100  next j
110  next i
120  for x=1 to 10:print w$(x) :next x
```

Es werden zuerst 10 Worte verlangt, die dann in des Text-Feld W$() eingetragen werden. Danach wird sortiert, indem in einer Schleife die Worte verglichen und gegebenenfalls durch den SWAP-Befehl vertauscht werden.

Die Texte in obigem Programm können nun beispielsweise Namen, Adressen oder auch Telefonnummern sein, die in einer Datei angeordnet sind. Man kann diese Sortierroutine auch für umfangreichere Textfelder verwenden, wie z.B. mehrdimensionale String-Arrays. Ein Beispiel für eine solche Anwendung ist leicht zu finden. Das Desktop kann ja auch die Einträge in einem Inhaltsverzeichnis der Diskette nach verschiedenen Kriterien sortieren, wie z.B. nach Namen, Typ oder Länge. Ebenso könnte ein BASIC-Programm eine Adressverwaltung vornehmen, in der die Einträge nach Namen, Adresse oder Telefonnummer sortiert ausgegeben werden. Die Eingabe für diese Daten würde noch besser aussehen, wenn wir sie als Maskeneingabe vornähmen.

7.4.1 Masken-Eingaben

Maskeneingaben werden in allen üblichen Datenverwaltungsprogrammen eingesetzt. Hier wird ein Formular auf dem Bildschirm dargestellt, in dem für die Einträge Platz gelassen wird. Der Bediener kann dann die Daten nur in die dafür vorgesehenen Bildpositionen eintragen. Der fertige Bildschirm sieht danach evtl. genauso aus wie das Formular, welches nachher auch gedruckt wird.

```
10   rem *** Maskeneingabe ***
20   dim t$(100,6),x(7),y(7)   : rem Felder definieren
30   fullw 2 : clearw 2        : rem Fenster vorbereiten
40   for i=1 to 7
50   read x(i),y(i),a$         : rem Maske lesen
60   gotoxy x(i),y(i)          : rem Cursor setzen
70   print a$;left$(".........",11- len(a$));" :"
80   next i
90   data 10,1,"** Adressen eingeben **"
```

```
100 data 1,5,"Name"
110 data 16,5,"Vorname"
120 data 1,10,"Ort"
130 data 1,12,"Straße"
140 data 1,15,"Telefon"
150 data 2,18,"Bemerkungen"
160 n=1                           : rem erster Eintrag
165 gotoxy 1,1 :print n;" )" : rem Nummer schreiben
170 for i=1 to 6
180 gotoxy x(i+1)+6,y(i+1)   : rem Cursor setzen
190 input t$(n,i)             : rem Eingabe
195 if t$(n,i)="#" then i=10 : rem Abbruch!
200 next i
210 if i<10 then n=n+1 : goto 165
220 n=n-1                     : rem Anzahl korrigieren
```

In diesem Programm wird eine Maske erstellt und auf den Bildschirm bzw. ins OUTPUT-Fenster gebracht. Danach werden die Eingaben selbst vorgenommen. Gibt man in irgendeinem Eintrag ein #-Zeichen ein, wird die Eingabe abgebrochen. In den folgenden Zeilen kann dann mit den eingegebenen Daten weitergearbeitet werden, wobei in T$(n,1) der n-te Name, in T$(n,2) der nte Vorname usw. steht. Die Variable N enthält die Anzahl der Einträge, die insgesamt vorgenommen wurden.

Nun kann auch die Sortierroutine von vorhin eingesetzt werden. Will man nach den Telefonnummern sortieren, so wird lediglich nach T$(n,5) sortiert.

7.5 Steuerungen: Maus/Joystick

Um den Ablauf eines Programmes zu beeinflußen, wird im allgemeinen die Tastatur verwendet. Bei manchen Anwendungen kann es jedoch lästig sein, nur für eine Eingabe die Tastatur zu bedienen. Dies ist z.B. bei einem Spiel so, wenn es mit einem Joystick gespielt wird. Bei einer Abfrage, ob ein neues Spiel begonnen werden soll oder nicht, wäre es angenehm, wenn auch das mit dem Joystick zu erledigen wäre.

Eine solche Steuerung ist nicht schwer zu programmieren. Eine Ja/Nein-Abfrage kann durch Rechts- bzw. Linksbewegen des Joysticks oder auch durch die Betätigung der rechten oder linken Maustaste ausgeführt werden. Komplizierter wird es, wenn mehrere Auswahlmöglichkeiten existieren. Hier kann man ein Menü ausgeben, dessen Bedienung mit der Maus bzw. dem Joystick möglich ist. Hier ein Beispiel für eine solche Auswahltechnik:

```
10   rem ** Auswahl mit Joystick 0 **
15   def seg=1                   : rem PEEK/POKE Byte
20   clearw 2 : gotoxy 0,1       : rem Fenster vorbereiten
30   print "  1.":print "  2.":print "  3." : rem Menü darstellen...
40   y=peek(3582)                : rem vertikal Joystick
45   poke 3582,0                 : rem zurücksetzen
50   w=w-(y=1)+(y>1)-(w>3)+(w<1) : rem neue Position
60   gotoxy 0,w : print "=>"     : rem Wahl anzeigen
70   for i=1 to 100 : next i     : rem Pause
80   gotoxy 0,w : print "  "     : rem Zeiger löschen
90   if (peek(3581)and 3)=0 then 40
```

Hier wird zwischen drei Auswahlpunkten ein Zeiger hoch und runtergeschoben und mit dem Feuerknopf quittiert. Die Variable W enthält dann die Nummer des ausgewählten Menüpunktes. Dabei können natürlich beliebig viele Auswahlpunkte bearbeitet werden, sofern sie auf den Bildschirm passen. Hier nun die Version für die Maus:

```
10   rem ** Auswahl mit Maus **
20   clearw 2 : gotoxy 0,1       : rem Fenster vorbereiten
30   print "  1.":print "  2.":print "  3."  : rem Menü ausgeben
40   y=peek(9954)                : rem Mausposition
50   w=1-(y>70)-(y>90)           : rem Wahl bestimmen
60   gotoxy 0,w : print "=>"     : rem und anzeigen
70   for i=1 to 100 : next i     : rem Pause
80   gotoxy 0,w : print "  "     : rem Zeiger löschen
90   if (peek(3581)and 3)=0 then 40
```

Dieses Programm berechnet die ausgewählte Position für den Pfeil aus der vertikalen Position des Mauszeigers. Quittiert wird hier mit einem der beiden Maustasten. Die Darstellung des Auswahlzeigers ist eigentlich nicht nötig, da auch der Mauszeiger selbst genügt. Ein solcher Pfeil ist aber sicherer, um nicht durch kleine Ungenauigkeiten in der Auswertung falsche Ergebnisse zu erhalten.

Die Position des Mauszeigers kann natürlich auch zweidimensional ausgewertet werden. Die horizontale Position der Maus ist unter normalen Bedingungen in der Speicherzelle 9952 zu finden, der vertikale Standort in 9954. Diese beiden Speicherworte lassen sich mit POKE auch manipulieren und setzen den Mauszeiger an eine beliebige Position auf dem Bildschirm. Um den Mauszeiger in die linke obere Ecke des Bildschirms zu bringen, braucht man also nur POKE 9952,0 und POKE 9954,0 zu tippen bzw. zu programmieren. Dies ist interessant, wenn man z.B. eine Zeichnung mit der Maus anfertigen will und dafür eine definierte Anfangsposition benötigt.

7.6 Ein-/Ausgaben

Nachdem wir den Computer im Speicher und auf dem Bildschirm allerhand machen ließen, ist es nun an der Reihe, ihn auch mit externen Geräten arbeiten zu lassen. Dafür sind die Diskettenstation und der Drucker zuständig.

Die normalen Ein- und Ausgaben auf diese Geräte sind bereits besprochen worden. Nun sind die Möglichkeiten dieser Peripheriegeräte noch nicht vollständig ausgeschöpft. Die Disketten sind in Sektoren eingeteilt, deren einzelne Zugriffe nicht ohne weiteres möglich sind. Beim Drucker gibt es ebenfalls noch Funktionen, deren wir uns annehmen wollen. Und zum Abschluß werden wir uns die Bedienung und Programmierung eines Telefonmodems ansehen.

Die Anschlüsse der einzelnen Schnittstellen sind ja bereits bekannt. So können wir uns um die Programmierung der angeschlossenen Geräte kümmern, und eventuell andere Anschlüsse herstellen.

7.6.1 Druckeransteuerung

Um irgendetwas auszudrucken, kann bekanntermaßen der LPRINT- bzw. LLIST-Befehl verwendet werden. Zusätzlich haben wir die Möglichkeit der Ausgabe mittels der OUT 0,x-Anweisung, durch die ein beliebiges Zeichen ausgegeben werden kann. Das beschränkt sich nicht auf die Ausgabe von Buchstaben, auch Steuerzeichen können so an den Drucker ausgegeben werden. Der Befehl OUT 0,x entspricht eigentlich genau der Anweisung LPRINT CHR$(x);, ist aber etwas einfacher zu schreiben und außerdem schneller.

Wenn man nun die Alternate- und die Help-Taste drückt, so sendet der Computer beim Ausdruck der Hardcopy etliche Steuerzeichen. Diese steuern einerseits den Druckermodus, damit er die gesendeten Werte nicht als ASCII-Werte interpretiert und Unsinn druckt. Andererseits wird der Drucker angewiesen, die Zeilenvorschübe einzustellen, damit nicht jede Graphikzeile einen Abstand zur vorhergehenden Zeile hat. Das Problem, welches sich dabei ergeben kann, ist die Inkompatibilität einiger Drucker. Das bedeutet, daß einige Drucker andere Steuerzeichen für die Funktionen besitzen. Die Anpassung der Druckerschnittstelle, die das Desktop bereitstellt, ist hier keine Hilfe.

Das von der Diskette geladene Betriebssystem nimmt die benötigten Steuerzeichen aus einer Tabelle, die im Speicher ab $17A5E liegt. Diese Tabelle läßt sich somit natürlich manipulieren, um einen anderen Drucker an den ST anzupassen. Die Reihenfolge der Einträge der Tabelle ist folgende:

$17A5E	Esc "X" 6	
$17A63	Esc "X" 5	
$17A68	Esc "X" 3	für ATARI Farb-Matrix-Drucker
$17A6D	Esc "L"	S/W-Matrixdrucker: 960 Punkte/Zeile
$17A71	Esc "Y"	Farbdrucker: 960 Punkte/Zeile
$17A75	Esc "3" 1	1/216 Zoll Zeilenabstand
$17A7A	Esc "3" 1	s.o.
$17A7F	Esc "1"	7/72 Zoll Zeilenabstand
$17A74	Esc "3" 1	s.o.
$17A83	Esc "2"	1/6 Zoll Zeilenabstand
$17A88	Esc "x" 0	

Escape L stellt dabei z.B. einen EPSON-S/W-Drucker auf den Empfang von direkt zu druckenden Bitmustern ein. Diesem Zeichen folgt dann die Anzahl der zu sendenden Bytes für die Zeile. Danach wird das Bitmuster jedes weiteren Bytes untereinander in die aktuelle Spalte gedruckt. Hat Ihr Drucker für diese Einstellung oder eine der anderen ein von obiger Tabelle abweichendes Steuerzeichen, so können Sie ihn durch das EinPOKEn des neuen Zeichens anpassen.

Der Ausdruck des Bildschirms kann außer durch den kombinierten Tastendruck auch durch Anwahl des Menüpunktes *Bildschirm ausdrucken* oder durch den Befehl POKE 1262,0 starten. Den Ausdruck übernimmt dann das Betriebssystem vollständig, solange nicht noch einmal Alternate und Help gedrückt wird.

Um die Programmierung des Druckers selbst vorzunehmen, um z.B. eigene Symbole auszudrucken, muß man die entsprechenden Befehlssequenzen selbst dem Drucker übermitteln. Dabei hat man zusätzlich die Wahl der Auflösung, in der der Drucker arbeiten soll. Escape L stellt diese auf doppelte Dichte ein, was für einen sauberen Ausdruck des schließlich hochauflösenden Bildschirms nötig ist. Ebenso kann man, jedenfalls bei einem EPSON-S/W-Drucker bzw. einem kompatiblen, mit Escape K die normale Auflösung wählen. Ein kleines Beispielprogramm für die Ausgabe eines eigenen Zeichens:

```
10   rem ** eigenes Zeichen drucken **
20    for i=1 to 16
30   read a          : rem Zeichen lesen
40   out 0,a         : rem und ausgeben
50   next i
60   lprint          : rem fertig, Zeilenvorschub
70   data 27,75,12,128
80   data 4,10,26,58,103,231
90   data 231,103,58,26,10,4
```

Dieses Beispiel gibt ein kleines UFO auf dem Drucker aus. Die 12 ist dabei die Anzahl der Bilddaten. Bei einer größeren Anzahl als 255 muß das High-Byte der Anzahl zu der 128 addiert werden. Danach folgen die Daten selbst, deren niederwertigstes Bit unten gedruckt wird.

7.6.2 Diskettenverwendung

Um die Disketten als Speichermedium für unsere Programmdaten zu verwenden, können wir die LOAD- und SAVE-Befehle oder auch BLOAD und BSAVE verwenden, wie bereits in Kapitel 2.2 beschrieben. Die gesamte Steuerung des Diskettenzugriffes übernimmt dann das BASIC bzw. das GEMDOS. Nun ist es auch möglich, diese Steuerung selbst in die Hand zu nehmen und das Betriebssystem zu umgehen. Hierfür brauchen wir ein Maschinenprogramm, welches wieder in einer Textvariablen abgelegt werden kann.

In diesem Abschnitt wollen wir ein solches Programm betrachten, welches aber auch noch mehr kann. Der erste Teil des Programmes liest das Maschinenprogramm aus DATA-Zeilen in die Textvariable ein. Dieses Verfahren haben wir ja schon mehrmals angewendet, so daß es keiner weiteren Erläuterung bedarf.

Der zweite Teil des Programmes besteht aus der Abfrage, welcher Sektor der Diskette gelesen werden soll. Der eingegebene Wert wird dann in das Maschinenprogramm eingesetzt, um es für diesen einen Sektor vorzubereiten. Die Textvariablen

D$(0) und D$(1) werden danach für die Aufnahme der Daten aus dem Sektor vorbereitet. Hier werden leider zwei hintereinanderliegende Variablen benötigt, da ein String nur maximal 255 Bytes lang sein kann. Die Variable D wird daraufhin mit der Anfangsadresse des Puffers geladen.

Nun folgt der Aufruf des Maschinenprogrammes, bei dem als Parameter die Pufferadresse mit übergeben werden muß. Die Diskettenstation läuft an und der Inhalt des gewählten Sektors wird in den Puffer geladen. Da jeder Sektor immer 512 Bytes enthält, werden D$(0) und D$(1) teilweise überschrieben. Es kann hier natürlich auch ein anderer Datenpuffer verwendet werden, wobei man sich jedoch genau überlegen sollte, ob dieser Speicherbereich auch wirklich von keinem anderen Programm verwendet wird.

Ist der Sektor nun in den Speicher geladen worden, so wird sein Inhalt ab Zeile 200 hexadezimal und charaktermäßig dargestellt. Dies ist interessant, wenn man den Aufbau des Directory, also des Inhaltsverzeichnisses, auf der Diskette erforschen will. Dieses Inhaltsverzeichnis beginnt beim ATARI-ST-Format bei einseitig formatierten Disketten in Seite 0, Track 1, Sektor 3.

```
10   rem *** Disketten - Zugriff ***
20   fullw 2                    : rem Output-Fenster groß
30   def seg=0                  : rem Wortzugriff setzen
40   a$=space$(42)              : rem String vorbereiten
50   b=peek(varptr(a$)+2)       : rem Adresse feststellen
60   for i=0 to 40 step 2
70   read a                     : rem Daten lesen
80   poke b+i,a                 : rem und einsetzen
90   next i
95   rem --- Maschinenprogramm-Daten ---
100  data 8815,14,16188,1,16188,0
110  data 16188,1,16188,6,16188,0,17063,12041
120  data 16188,8,20046,-8196,0,20,20085

130  print :input "Seite, Track, Sektor";x,y,z
140  poke b+10,x                : rem Sektornummer
```

```
142  poke b+14,y                   : rem Spurnummer
144  poke b+18,z                   : rem Diskettenseite
150  dim d$(4)                     : rem Puffer vorbereiten
160  d$(0)=space$(255)
170  d$(1)=space$(255)
180  d=peek(varptr(d$(0))+2)       : rem Adresse feststellen

190  call b (d)                    : rem Sektor laden!

200  def seg=1                     : rem Byte-Zugriff
210  print :for i = 1 to 512
220  x% = peek(i+d-2)              : rem Byte lesen
230  if x%<16 then print "0";      : rem nur für's Format
240  print hex$(x%);" ";           : rem Hex-Byte schreiben
245  if (i and 15) >0 then 300: rem Zeile fertig ?
250  print "  "; :for j = 17 to 2 step -1
260  x%=peek(i+d-j)
270  if x%<11 then ?"."; : goto 290
280  print chr$(x%);               : rem ASCII-Zeichen
290  next j : print                : rem nächstes Zeichen
300  next i                        : rem nächste Zeile
```

Das Maschinenprogramm sieht hier recht simpel aus:

```
MOVE.L 14(SP),a1        * Pufferadresse retten
MOVE.W #1,-(SP)         * Anzahl der Sektoren =1
MOVE.W #0,-(SP)         * Seite 0
MOVE.W #1,-(SP)         * Spur =1
MOVE.W #6,-(SP)         * Sektor =6
CLR.L     -(SP)         * ? (muss sein)
MOVE.L a1,-(SP)         * Pufferadresse
MOVE.W #8,-(SP)         * Kommando Floppy-Read
TRAP #14                * XBIOS-Aufruf
ADDQ.L #14,SP           * Stack reparieren
RTS                     * zurück zum BASIC
```

Hier läßt sich nicht nur die Sektornummer einstellen, sondern auch das Laufwerk, die Anzahl der zu ladenden Sektoren und die Datenrichtung. Es kann nämlich auch aus dem Puffer in den Sektor geschrieben werden, indem man aus der 0 bei Datenrichtung eine 1 macht. Doch Vorsicht: solche Schreibversuche können bei einem falschen Sektorinhalt die Diskette völlig umkrempeln, wenn man z.B. das Inhaltsverzeichnis überschreibt. Deshalb ist es ratsam, vorerst mit einer Diskette zu arbeiten, auf der nichts Wichtiges gespeichert ist.

Dies war also eine Möglichkeit, einen direkten Diskettenzugriff vorzunehmen. Der normale Weg ist aber die Wahl eines Namens, unter dem die gewünschten Daten auf der Diskette abgelegt sind. Man gibt dann ein LOAD "Pfad:Name.Typ", und das Programm wird geladen. Der Pfad bedeutet hier die Angabe des zu verwendenden Laufwerks und/oder die Bezeichnung des Unterdirektories, des Ordners, in dem die Datei zu finden ist. Der Name ist die Bezeichnung der Datei selbst und kann maximal 8 Zeichen lang sein. Der Typ, der am Schluß der Anweisung steht, ist ein aus drei Buchstaben bestehendes Kürzel für die Art der Datei. Dieser Typ ist eigentlich frei wählbar, jedoch sind einige Typenbezeichnungen reserviert. PRG bedeutet, daß es sich um ein lauffähiges Maschinenprogramm handelt, BAS wird für BASIC-Programme eingetragen usw..

Diese Typenbezeichnungen helfen bei der Auswahl eines Programmes bzw. einer Datei. Sucht man die BASIC-Programme auf einer Diskette, so kann man mit DIR *.BAS die Liste der verfügbaren Programme bekommen.

Genau so verfährt auch die Dateiauswahl, die erscheint, wenn man die Ladeoder Abspeicherungsfunktion aus dem Menü auswählt. Es entsteht ein Fenster auf dem Bildschirm, das in mehrere Einzelfenster unterteilt ist. Betrachten wir mal die oberste Zeile. Hier stehen die Auswahlkriterien und die Pfadbezeichnung, die die zur Verfügung stehenden Dateien auswählen. Enthält die Zeile also *.BAS, so werden alle Programme mit dem Typenzusatz (Extender) .BAS ausgewählt. Das Sternchen steht dabei stellvertretend für alle anderen Zeichen, die auftreten können. Durch Anklicken dieser Zeile kann man nun

diese Auswahlzeile ändern, um z.B. die Programme einer anderen Diskettenstation anzeigen zu lassen.

Dieses gesamte Fenster mit allen Bedienungsmerkmalen wird vollständig vom GEM-AES bearbeitet. Wir können somit ohne viel Aufwand diese Dateienauswahl für ein BASIC-Programm verwenden, wenn wir z.B. eine Dateiverwaltung geschrieben haben und nun eine Datei dafür von der Diskette laden wollen. Um dann die Funktion aufzurufen, die uns das Auswahlfenster zeichnet und die Bedienung übernimmt, müssen wir einige Parameter vorbereiten.

Wir benötigen dafür erst einmal zwei Puffer. In dem einen wird das vorhin besprochene Auswahlkriterium abgelegt, der andere enthält einen vorgegebenen Dateinamen. Diese Vorgabe ist nicht notwendig, aber in dem Puffer wird nach Anwahl des OK- oder Abbruch-Knopfes der gewählte Dateiname abgelegt. In Verbindung mit der Pfaddefinition des anderen Puffers haben wir dann die genaue Bezeichnung der Datei zur Verfügung, die wir nun für die Abspeicherung bzw. für das Laden unserer Daten verwenden können. Hier nun ein Programmvorschlag, der diese Funktion aufruft:

```
10   rem *** File-Select ***
20   poke contrl,122           : rem VDI-Aufruf zum
30   poke intin,0
40   vdisys 0                  : rem Cursor einschalten

50   defdbl b,d : b=gb
60   cn=peek(b)
70   d=peek(b+16)              : rem Feld-Definitionen
80   io=peek(b+12)
90   a$="\*.PRG"               : rem Pfad definieren
100  for i=1 to 40 : a$=a$+chr$(0) : next
110  for i=1 to 15 : b$=b$+chr$(0) : next
120  a = peek(varptr(a$)+2)
130  c = peek(varptr(b$)+2)
```

```
140 poke cn,90          : rem AES-Kommando
150 poke cn+2,0
160 poke cn+4,2
170 poke cn+6,2
180 poke cn+8,0
190 poke d,a            : rem Pfad setzen
200 poke d+4 ,c         : rem Namenpuffer
210 gemsys 90           : rem Aufruf
220 print "In ";a$
230 print b$;" gewählt"    : rem Ergebnis-Ausgabe
```

Der Anfang dieses Programmes besteht aus einem VDI-Aufruf. Diese Funktion schaltet den Mauszeiger an, der ja bei einem Tastendruck verschwindet. Läßt man die VDI-Funktion weg und startet das Programm, ohne die Maus dabei zu bewegen, so bleibt der Cursor während des gesamten Funktionsablaufes unsichtbar. Das ist aber ein gewisses Handicap für die Programmauswahl...

Im Anschluß daran werden nun die Zeiger für die AES-Parameter definiert. Die Erklärung dieses Vorganges finden Sie in dem Kapitel über Menü- und Fensterprogrammierung. Des weiteren werden nun die beiden Puffer für die Dateidefinition vorbereitet. Die beiden Schleifen füllen diese Puffer, bei denen es sich wieder um Textvariablen handelt, mit dem Wert 0 auf. Das ist für die richtige Funktion der AES-Routine nötig.

Als nächstes werden die Parameter in die Parametertabellen des AES übertragen. ADDRIN (D) erhält dabei die beiden Adressen der Puffer als Langworte. Nun schließlich erfolgt der eigentliche Aufruf des GEM-AES. Nach Anklicken einer der beiden Auswahlknöpfe oder einem Doppelklick auf einem Dateinamen wird das Auswahlfenster wieder gelöscht und die Kontrolle an das aufrufende BASIC-Programm übergeben. Die beiden Textvariablen enthalten nun die Pfad bzw. Namenangabe der ausgewählten Datei.

Zusätzlich übergibt die Funktion noch die Angabe, durch welchen Knopf (OK oder Abbruch) die Beendigung der Funktion erfolgt ist. Diese Information wird in INTOUT(1) angeboten. Leider jedoch zerstört das BASIC diese Information, bevor wir sie auslesen können, so daß wir dafür die Position der Maus auswerten müssen.

7.6.3 Kommunikation

Die Datenfernübertragung (DFÜ) über die Telefonleitung hat sich in der letzten Zeit zu einem weitverbreiteten Hobby von Computerbesitzern entwickelt. Dabei spielt weniger das Hacken in fremden Computersystemen als die Kommunikation per Mailbox eine Rolle. In Deutschland existieren bereits jetzt eine unüberschaubare Anzahl solcher öffentlichen Mailboxen, daß die Qual der Wahl groß ist. Doch wollen wir erst einmal betrachten, wie wir mit dem ATARI ST in die Welt der Telekommunikation einsteigen können.

Zunächst einmal benötigen wir ein Telefonmodem. Solche Geräte, die die Verbindung zwischen Telefon und Computer herstellen, werden im Handel angeboten. Man sollte bei der Auswahl eines solchen Modems allerdings darauf achten, daß das Gerät auch von der deutschen Bundespost genehmigt ist, da der Betrieb anderer Modems verboten ist.

Ein handelsübliches Modem besitzt meistens einen seriellen Anschluß für die Verbindung zum Computer und zwei Muscheln, in die der Telefonhörer nach Aufbau der Verbindung gelegt wird. Die Datenübertragung erfolgt dann durch einen nerventötenden Pfeifton, dessen Auswertung die im Modem eingebaute Elektronik übernimmt. Die Geschwindigkeit dieser Übertragung ist bei Verwendung akustisch gekoppelter Modems auf 300 Baud festgelegt, da bei einer höheren Rate die Datensicherheit nicht mehr gewährleistet werden kann. Die von der Post angebotenen Modems, die galvanisch, also direkt an das Telefonnetz, angeschlossen sind, können auch Übertragungsraten bis zu 1200 Baud verarbeiten.

Aus diesem Grund stellen wir die serielle Schnittstelle erst einmal auf 300 Baud ein. Da die meisten Mailboxen jedes empfangene Zeichen direkt zurücksenden, Echoen genannt, sollten wir auch die Betriebsart Voll-Duplex wählen.

Wir schließen nun das Modem an den ebenso bezeichneten Stecker des ST an. Um nun mit der DFÜ beginnen zu können, müssen wir ein geeignetes Programm starten, welches die Übertragung abwickelt. Die einfachste Methode ist, den Terminal-Emulator des ATARI ST aufzurufen. Jedes Zeichen, das man dort eintippt, wird automatisch an der seriellen Schnittstelle ausgegeben, sowie alle empfangenen Zeichen direkt auf dem Bildschirm sichtbar werden. Wir können daher sofort eine Mailbox anrufen, den Hörer auf das Modem legen und über Tastatur und Bildschirm mit der Gegenseite kommunizieren.

Die Sache hat aber einen Haken: Wir haben bei der Verwendung des Terminal-Emulators keine Möglichkeit, unser Gespräch mit der Gegenseite auszudrucken oder abzuspeichern. Da manche Mailboxen lange Einleitungstexte oder Hilfsmenüs senden, kann so manche Information verloren gehen, wenn man nicht so schnell lesen kann, wie die Zeilen oben verschwinden.

Also müssen wir auf den Komfort des Emulators verzichten und selbst ein kleines Programm schreiben, welches die Kommunikation per Modem unterstützt und gleichzeitig die empfangenen Texte speichert. Ein solches Programm muß ständig die Schnittstelle und die Tastatur abfragen, ob ein Zeichen vorliegt. Auf diese Weise kann der Duplex-Betrieb realisiert werden, bei dem Senden und Empfangen beinahe gleichzeitig ablaufen kann.

Nun ergab sich bei der Programmerstellung ein Problem. Die INKEY$-Funktion, mit der man üblicherweise die Tastatur abfragen kann, ohne daß das Programm stoppt, funktioniert in der vorliegenden BASIC-Version nicht. Nimmt man stattdessen den INP(2)-Befehl, so wartet das Programm ab, bis eine Taste gedrückt wurde. In dieser Zeit könnte jedoch ein Text empfangen werden, den man daher nicht lesen könnte!

Wir müssen diesen Befehl also irgendwie nachbilden. Dafür können wir einen Trick anwenden, der mit PEEK und POKE realisierbar ist. Hier ein Beispielprogramm:

```
10   rem ** INKEY$-Simulation **
20   p=3510                      : rem Zeiger auf Zeichen
30   x=peek(p)                   : rem zwischenspeichern
40   gotoxy 1,1 : print a        : rem Beispiel
45   a=a+1
50   if peek(p)=x then 30        : rem neues Zeichen?
60   x=peek(p)                   : rem ja, neuer Zeiger
70   t=peek(3086+x) and 255      : rem Zeichen holen
80   print chr$(t)               : rem und ausgeben
90   goto 40                     : rem Endlos-Schleife
```

Jedes in die Tastatur eingetippte Zeichen wird vom Betriebssystem automatisch in einem Puffer abgelegt. Dieser Puffer beginnt in Speicherzelle 3086. Die Information, welches Zeichen im Puffer gerade aktuell ist, steht in 3510. Diesen Zeiger lesen wir in Zeile 30 aus und speichern ihn in der Variablen X ab. Wird nun ein weiteres Zeichen eingegeben, so wird dieser Zeiger um 2 erhöht. Diese Veränderung wird in Zeile 50 festgestellt. Der neue Zeiger wird nun ausgelesen und zum Pufferbeginn addiert. Dort liegt das neue Zeichen, welches wir nun durch PEEK(3086+X) erhalten. Das Zeichen wird nun ausgegeben, und das eigentliche Programm weitergeführt. An diesem Beispiel sieht man deutlich, daß das Programm nicht auf einen Tastendruck wartet, sondern ständig läuft.

Diesen Trick verwenden wir nun in unserem Terminal-Programm, welches dadurch auch den Duplexbetrieb beherrscht.

```
10   rem ** Terminal-Programm **
20   p=3510
30   dim a%(5000)                : rem Speicher reservieren
40   i=0                         : rem Zähler auf Null
50   z=peek(p)                   : rem Pufferzeiger lesen
```

```
60   if inp(-1)=0 then 100   : rem Zeichen empfangen?
70   I=I+1                   : rem ja, Zähler erhöhen
80   a%(i)=inp(1)            : rem Zeichen abspeichern
90   print chr$(a%(i));      : rem und ausgeben

100  if peek(p)=z then 60    : rem Taste gedrückt?
110  z=peek(p)               : rem ja, neuen Zeiger lesen
120  x=peek(3086+z) and 255  : rem und Zeichen feststellen
130  if x=13 then print      : rem evtl. Zeilenvorschub
140  print chr$(x);          : rem Zeichen ausgeben,
150  out 1,x                 : rem senden
160  i=i+1 : a%(i)=x         : rem und abspeichern
170  if x<> asc("*") then 60  : rem Ende?

180  for j=1 to i            : rem ja, alles ausgeben
190  print chr$(a%(j));
200  next j
```

In den Zeilen 10 bis 50 werden einige Vorbereitungen getroffen. Zeile 60 fragt die serielle Schnittstelle ab, ob ein Zeichen empfangen wurde. Wenn ja, wird dieses Zeichen in dem A%-Feld abgespeichert und auf dem Bildschirm ausgegeben.

Die Zeilen 100 bis 160 testen den Tastaturbuffer und verarbeiten ein eingegebenes Zeichen. Auch diese Zeichen werden in dem A%-Feld abgespeichert, damit man später die gesamte Kommunikation nachvollziehen kann. Arbeitet die Gegenstation jedoch im Echo-Modus, so können die Zeilen 140 und 160 entfallen, da die eingegebenen Zeichen ohnehin zurückgesendet und dabei angezeigt und abgespeichert werden.

Gibt man nun ein Sternchen (*) ein, so wird das Programm abgebrochen und der gespeicherte Text vollständig ausgegeben. An dieser Stelle kann statt des PRINT natürlich auch ein LPRINT stehen, was die Ausgabe auf den Drucker umleitet und das gesamte Gespräch auf Papier bringt.

7.7 Zeichensatzänderung

Der Zeichensatz des ATARI ST umfasst eine enorme Vielfalt von Zeichen und Symbolen. So sind die griechischen Buchstaben vorhanden, was für Physiker sehr interessant ist. Ebenso sind mathematische Symbole wie Wurzel, Integral oder auch 1/2 dabei. Man kann somit wissenschaftliche Texte auf dem Bildschirm gleich mit den entsprechenden Formeln versehen.

Für einige Anwendungen wäre aber auch angebracht, eigene Symbole direkt als ein Charakterzeichen zu definieren. Die graphische Umgebung von Spielen wird damit zu einer einfachen PRINT CHR$()-Folge, was die Geschwindigkeit des Bildaufbaus erhöhen kann und gleichzeitig die Programmierung erleichtert.

Nun bietet das BASIC hierfür keine Funktion. Man muß also wieder einmal auf die PEEK- und POKE-Befehle zurückgreifen und den Zeichensatz direkt ändern.

Im Betriebssystem sind 3 Zeichensätze bereits eingebaut. Diese sind von den Symbolen her gleich, besitzen jedoch unterschiedliche Auflösungen. Für die Darstellung auf dem Monochrome-Monitor ist ein 8*16-Zeichensatz vorgesehen, d.h. jedes Zeichen ist insgesamt 8 Punkte breit und 16 Punkte hoch. In Farbe sind es nur noch 8*8, da die vertikale Auflösung nur noch halb so groß ist. Und schließlich ist noch ein 6*6-Font vorhanden für die Beschriftung der Bilder, die in der entsprechenden Darstellung die Dateien im Inhaltsverzeichnis beschriften. Font ist übrigens der gebräuchliche Ausdruck für einen Zeichensatz.

Um nun einen dieser Zeichensätze zu manipulieren, muß man zuerst wissen, wo dieser im Speicher liegt. In der TOS-Version, die von Diskette geladen wird, liegt der Anfang des 8*16-Fonts an der Adresse 104536. Ab hier sind die ersten 256 Byte der oberen Ebene jedes der 256 Zeichen im Font abgelegt, danach folgen die Daten der nächsten Ebene und so weiter. Das höchstwertigste Bit des ersten Bytes eines Zeichens bestimmt also den oberen, linken Punkt in der Darstellung dieses Zeichens. In dem normalen Zeichensatz ist die obere Ebene immer 0, also weiß.

Das verhindert, daß zwei übereinander stehende Zeichen sich berühren, was das Schriftbild verschlechtern würde.

Wir wollen nun ein Programm betrachten, das das Zeichen, welches PRINT CHR$(1) ausgibt, verändert. Dieses Zeichen steht in einem Graphikfenster in der oberen, rechten Ecke und stellt das Verschiebesymbol des Fensters dar. Man erhält das Zeichen auch durch Tippen von Control-A.

Das Programm verwendet wieder die Binär-Dezimal-Umwandlung, die wir bereits kennen. Der Unterschied zu dem Programm zur Definition des Füllmusters aus Kapitel 7.1.5 ist der, daß hier nur 8 Bit verarbeitet werden. Sonst ist es das gleiche Prinzip wie gehabt. Als neues Zeichen wird wieder ein Männlein eingesetzt, was natürlich auch hier beliebig veränderbar ist.

Achtung: Dieses Programm funktioniert nur, wenn das TOS von Diskette geladen wird!

```
10   rem *** Fontzeichen verändern ***
20   def seg=1                 : rem Bytebreite einstellen
80   for i=0 to 12 : read x$ : rem Füllmuster
82   x=0 : for j=1 to 8        : rem Umwandlung
84   x= x-(mid$(x$,j,1)<>" ") * 2^(8-j)
86   next j
90   poke 104536+i*256,x       : rem Muster setzen
100  next i
110  print chr$(1)             : rem Probeausgabe
120  end
130  rem +++ Musterdaten +++
150  data "   ***  "
160  data "* * * * "
170  data "*  ***  "
180  data "**  *   "
190  data " *******"
200  data "  ******"
210  data "   *** *"
220  data "  ******"
230  data "  ** ** "
```

```
240  data "  ** ** "
250  data "  ** ** "
260  data "  ** ** "
270  data " *** ***"
```

Will man ein anderes Zeichen verändern, so braucht man nur in Zeile 90 die 104536 zu verändern. Für die Gestaltung eines neuen A, also CHR$(65), müßte man nur 64 addieren, da 104536 das Zeichen Nummer 1 adressiert. Somit können Sie also Ihren eigenen Zeichensatz kreieren, den Sie dann mit BSAVE "MYFONT.FNT",104536,4096 auf der Diskette ablegen und später mit BLOAD "MYFONT.FNT",104536 wieder laden können.

Der Dateiname ist in dem Beispiel willkürlich gewählt. Sie können mit entsprechend benannten Dateien beliebig viele Zeichensätze abspeichern und so für jede Anwendung einen eigenen Zeichensatz anlegen. So können Spiele unterstützt werden oder auch in kyrillischer Schrift Texte dargestellt werden. Den Anwendungsmöglichkeiten sind also fast keine Grenzen gesetzt.

Ein Rat noch: bevor Sie einen eigenen Zeichensatz erstellen oder laden, sollten Sie den Originalfont erst einmal abspeichern. Wenn Sie die neuen Zeichen nicht mehr brauchen, können Sie dann durch Einladen des Originalzeichensatzes sicherstellen, daß Sie das Inhaltsverzeichnis der Disketten oder die Menüpunkte des Desktop auch noch lesen können...

Nun gibt es noch eine weitere Möglichkeit, einen Zeichensatz zu variieren. Dafür müssen lediglich die Daten des Zeichensatzes in den RAM-Speicher kopiert und ein Zeiger des Betriebssystems verändert werden. Es handelt sich bei diesem Zeiger um denjenigen, welchen das TOS zum Zugriff auf die Font-Daten verwendet.

Diese Methode hat zwei große Vorteile:

1) Liegt der Original-Zeichensatz im ROM, so kann er nicht modifiziert werden. Die Daten im RAM jedoch können beliebig variiert und auch geladen bzw. abgespeichert werden.

2) Der Zeiger, der in dem folgenden Beispiel verwendet wird, ist nur für die Ausgabe mittels der OUT-Funktion relevant. Somit stehen zwei unterschiedliche Zeichensätze zur Verfügung, die einerseits durch PRINT und andererseits durch OUT 2,x ausgegeben werden können; beide Fonts können also auch gemischt ausgegeben werden.

Betrachten wir nun ein Beispiel für die Anwendung der beschriebenen Methode:

```
10   defdbl a,b,c          :'Zeiger vorbereiten
20   a=&H19858             :'Adresse des Original-Fonts
30   c=&H2924              :'Adresse des Zeigers
40   x=a                   :'Font-Zeiger kopieren
50   dim a%(2050)          :'Platz reservieren
60   for i=0 to 2050
70   a%(i)=peek(x+i*2)     :'Font-Daten kopieren
80   next
90   end
100  c$="A"                :'zu änderndes Zeichen
110  n=&H66                :'Bits der obersten Zeile
120  b=varptr(a%(0))       :'Adresse des neuen Fonts
130  poke c,b              :'Zeiger verbiegen
140  m=&HFF                :'Maske vorbereiten
150  o=asc(c$)
160  if (o mod 2)=1 then m=m*256 else n=n*256
170  a%(o/2)=a%(o/2) and m or n
180  gosub 300             :'Probetext modifiziert
190  poke c,a              :'Originalfont setzen
200  gosub 300             :'Probetext original
210  end
```

```
300   restore
310   for i=1 to 9        :'Probetext 'ABC' ausgeben
320   read x
330   out 2,x :next
340   return
350   data 27,72,27,66,27,66,65,66,67
```

Der erste Teil dieses Programmes kopiert nach Vorbereitung der benötigten Zeiger den Original-Font in das Integer-Feld a%. Dabei ist zu beachten, daß somit in jedem Eintrag dieses Feldes zwei Bytes des Zeichensatzes liegen. Aus diesem Grunde muß für den Zugriff auf ein bestimmtes Byte eine Maske verwendet werden, um das benachbarte Byte nicht zu beeinflussen. Diese Maske wird im zweiten Programmabschnitt in der Variablen m eingesetzt. Mit Hilfe dieser Maske wird nun das zu modifizierende Byte ausgeblendet und mit dem neuen Wert n überschrieben. Im Beispiel wird der Wert $66 eingesetzt, was aus dem A ein Ä macht.

Der dritte Teil des Beispiels gibt nun einen Probetext "ABC" aus, was aber nun als "ÄBC" erscheint. Nach dem Zurückschalten auf den Originalfont wird der Probetext wiederholt ausgegeben und erscheint nun als "ABC".

An diesem Beispiel zeigt sich, wie einfach sich ein neuer Zeichensatz anwenden läßt. Lediglich die Erstellung der neuen Font-Daten ist etwas kompliziert.

Doch da kann das folgende Programm Abhilfe schaffen. Es stellt einen recht simplen Zeichensatzeditor dar, der zwar nicht besonders schnell (BASIC!), jedoch dank des Einsatzes der Maus sehr einfach zu bedienen ist.

Beim Start des Programmes erscheint auf dem Schirm ein Rechteck mit 8 mal 16 Punkten bzw. Sternchen. Daneben wird das aktuelle Zeichen mehrmals ausgegeben, und zwar in der Form, die gerade entworfen wurde. Dadurch ist eine direkte Kontrolle des neuen Zeichens möglich.

Ein Sternchen in dieser Form entspricht einem gesetzten (schwarzen) Punkt in der Darstellung des Zeichens. Einen solchen winzigen Punkt nennt man Pixel, wobei ein Zeichen aus 8 x 16, also 128 Pixeln besteht.

Neben diesem Muster wird noch das ursprüngliche Zeichen angezeigt, welches ja mit einem PRINT-Befehl ungeändert darstellbar ist.

Außerdem sieht man unter dem Originalzeichen die Symbole "<" und ">". Klickt man eines dieser Symbole an, so wird das vorhergehende bzw. das nächste Zeichen angezeigt.

Die Variation eines Zeichens ist nun denkbar einfach. Wird ein Sternchen (Punkt gesetzt) angeklickt, so wird dieser durch einen Punkt ersetzt. Gleichzeitig wird dieses Pixel in den Musterzeichen verschwinden. Ebenso wird ein Pixel wieder gesetzt, indem man den entsprechenden Punkt anklickt.

```
10    '** Zeichensatz-Editor **
20    dim a%(2050)
30    defdbl a,b,c
40    a=&H19858: c=&H2924
50    d=a
60    for i=0 to 2050
70    a%(i)=peek(d+i*2)                :'Font kopieren
80    next
90    b=varptr(a%(0))                  :'Zeiger auf neue Daten
100   fullw 2 : clearw 2
110   poke c,b                         :'neuen Font einstellen
120   ch=65                            :'erstes Zeichen 'A'

130   poke contrl,123 : vdisys 0       :'Mauszeiger ausschalten
140   gotoxy 12,2
150   ?"Zeichen : ",chr$(ch)           :'Originalzeichen
160   gotoxy 12,4 : ?"< >"
170   if (ch mod 2)=0 then o=8 else o=0
180   gotoxy 0,0 : ?"________"
190   for i=0 to 15                    :'Feld zeichnen
```

```
200   gotoxy 0,i+1 : ?"|";
210   x=a%(ch/2+128*i)
220   for j=0 to 7
230   if (x and 2^(7-j+o))=0 then ?"."; else ?"*";
240   next j : ?"|"
250   next i
260   ?" --------"

270   out 2,27: out 2,asc("Y")
275   out 2,36: out 2,45             :'Cursor positionieren
280   for i=1 to 5 : out 2,ch        :'5 neue Zeichen
285   next
290   poke contrl,122: vdisys 0      :'Mauszeiger einschalten
300   mx=int(peek(9952)/8)-1
310   my=int((peek(9954))/16)-4      :'Mausposition ermitteln
320   if (peek(3581) and 1)=1 then 420 :'rechte Maustaste
330   if (peek(3581) and 2)=0 then 290 :'linke  Maustaste
340   if my=3 and mx=24 then ch=ch-1: goto 130
350   if my=3 and mx=27 then ch=ch+1: goto 130
360   if mx>8 then 300

370   x=a%(ch/2+128*my)
380   m=2^(7-mx+o)                   :'angeklicktes Bit
390   if (x and m)=0 then x=x or m
                    else x=x and (m xor &HFFFF)
400   a%(ch/2+128*my)=x              :'erneuern
410   goto 130

420   gotoxy 0,18
430   input "Dateiname :",f$         :'Font abspeichern
440   bsave f$,b,4100
```

In den Zeilen 420 bis 440 wird der neue Zeichensatz auf Diskette abgespeichert. So kann er bei Bedarf wieder geladen und durch Verbiegen des Zeigers $2924 aktiviert werden. Durch diese Methode steht der Verwendung exotischer Schrift- und Zeichensätze für Ihre Programme nichts mehr im Weg!

8. Anhang

8.1 Begriffserklärungen

Adresse

Die einzeln ansprechbaren Speicherzellen des Arbeitsspeichers werden durchnummeriert. Diese Platznummern ermöglichen den Zugriff (Lesen/Schreiben) auf den Inhalt der Speicherzelle und stellen dessen Adresse dar.

Anklicken

Auswahl eines GEM-Symbols (Icons) durch dessen Berührung mit dem Maus-Zeiger und Betätigung der Maus-Taste.

Applikation

Ein Anwenderprogramm, das direkt lauffähig ist, wird Applikation genannt.

ASCII (American Standard Code of Information Interchange)

Da Computer intern nur Binärzahlen verarbeiten können, muß auch jedem alphanumerischen Zeichen eine Binärzahl zugeordnet werden. Die am meisten verbreitete Zuordnungsvorschrift hierzu 7- oder 8-Bit-ASCII mit entsprechend 128 oder 256 Zeichen.

Assembler

Als Assembler bezeichnet man Programme, die in Mnemocode geschriebene Maschinenspracheprogramme in den Objektcode des entsprechenden Prozessors übersetzt.

Baudrate

Die Baudrate ist eine Geschwindigkeitsangabe für die serielle Datenübertragung über RS 232-Schnittstelle. So bedeutet 300 Baud eine Übertragungsgeschwindigkeit von 300 Bit pro Sekunde an. Beim ATARI ST sind über das Kontrollfeld 300-9600 Baud einstellbar.

Bit (Binary Digit)

Computer sind intern nur in der Lage, Binärzahlen zu verarbeiten, d.h. Zahlen aus den Binärziffern 0 und 1 bzw. LOW und HIGH Binärziffern (Bits) sind die kleinsten Informationseinheiten in der Computertechnik.

Bus

Ein Bus ist ein Leitungssystem zwischen den einzelnen Baugruppen eines Computers, z.B. zwischen CPU und Arbeitsspeicher der Datenbus, Adressbus und Steuerbus, oder zwischen mehreren Geräten, z.B. Computer und Drucker, Peripheriebus. Weiter unterscheidet man zwischen unidirektionalem (Datenübertragung in nur einer Richtung) und bidirektionalem Bus (Datenübertragung in beiden Richtungen).

Byte

Ein Byte ist eine achtstellige Binärzahl. Sie stellt in den meisten Mikro- und Mini-Computern die kleinste adressierbare Dateneinheit dar, auch wenn CPU und Speicher wie beim ATARI ST in Zwei-Byte-Worten (16 Bit) oder Zwei-Wort-Langworten (32 Bit) organisiert sind.

C (Programmiersprache)

C ist eine höhere Programmiersprache, die jedoch viele Eigenschaften von der Maschinensprache beinhaltet. Sie ist sehr schnell, maschinennah und verhältnismäßig einfach zu erlernen. Die Struktur der Sprache basiert auf ALGOL und PASCAL. In C geschriebene Programme sind leicht auf andere Computer übertragbar, wenn für das Zielgerät auch ein C-Compiler existiert. Dadurch wurde zum ersten Mal erreicht, daß Betriebssysteme (z.B. UNIX oder GEM), ohne neu geschrieben werden zu müssen, auf Rechnern mit unterschiedlicher CPU implementiert werden können.

Cartridge

Cartridges sind Einschub-Kassetten mit bis zu 128 KByte Festspeicher, die in den Schacht in der linken Seite des ATARI ST eingesteckt werden können. Sie enthalten Applikationen oder Betriebssystem-Erweiterungen.

Control-Panel (Kontroll-Feld)

Dialogfenster, in dem verschiedene Parameter eingestellt werden können, z.B. Farben, Uhrzeit, Tastaturklick usw.

Centronics-Schnittstelle

Diese Schnittstelle wurde von der Firma Centronics zum Anschluß von Druckern eingeführt und hat sich als Standard etabliert. Es handelt sich um eine unidirektionale, 8-Bit breite Parallel-Schnittstelle mit Handshake-Leitungen. Beim ATARI ST ist diese Schnittstelle bidirektional ausgeführt.

CPU (Central Processing Unit)

In Microcomputern wird mit CPU der Microprozessor bezeichnet. Im ATARI ST wird der MC 68000 von Motorola verwendet.

Cursor

Mit Cursor wird eine unterschiedlich geformte Marke (Pfeil, Fliege, Fadenkreuz oder Block) bezeichnet, die mit der Maus (Maus-Cursor) oder der Tastatur (Text-Cursor) auf dem Bildschirm bewegt werden und die nächste Eingabeposition markieren kann.

Debugger

Hierbei handelt es sich um ein Hilfsprogramm zur Fehlersuche und -Korrektur in assemblierten und compilierten Maschinenprogrammen. Dabei können Haltepunkte (Break-Points) gesetzt werden und Speicherinhalte gelesen und verändert werden. (SID)

Desktop (Schreibtisch)

Die Haupt-Bildschirmebene unter GEM, die Menüleiste, Diskettensymbole und Papierkorb enthält.

Dialogbox (Dialogfenster)

Mit Dialogbox wird ein interaktives Fenster bezeichnet, das Informationen an den Benutzer ausgibt und auf eine Antwort wartet.

DMA (Direct Memory Access)

DMA bedeutet direkten Speicherzugriff und bezeichnet die Fähigkeit von Peripherie-Bausteinen, ohne Mitwirkung der CPU Daten in den Arbeitsspeicher zu schreiben oder daraus zu lesen.

Disassembler

Hilfsprogramm, mit dem es möglich ist, ein Maschinenprogramm in den Mnemocode zurückzuübersetzen. Dadurch wird eine Fehlerkorrektur und Änderungen in Programmen erleichtert, von denen kein Listing existiert.

Duplex

Mit Duplex wird ein Datenübertragungsverfahren bezeichnet, bei dem es möglich ist, Daten zur gleichen Zeit in beiden Richtungen zu transportieren (Vollduplex) bzw. kurzzeitig die Richtung wechselnd zu übertragen.

Emulation

Mit Emulation bezeichnet man den Vorgang, auf einem Gerät ein anderes mittels Software und/oder zusätzlicher Hardware zu simulieren. So wird z.B. auf dem ATARI ST im Terminal-Modus ein VT52-Terminal emuliert.

Festplatte (Hard-Disk)

Die Festplatte ist ein Speichermedium, das nach dem gleichen magnetischen Verfahren arbeitet wie Disketten. Im Unterschied zu diesen ist die Kapazität der Festplatten-Laufwerke jedoch um 1 bis 2 Größenordnungen (5 bis 500 MByte) und die Übertragungsraten um eine Größenordnung höher. Dies wird dadurch erreicht, daß die Diskette fest installiert ist und mit einer wesentlich höheren Drehzahl arbeitet (ca. 10-fach).

File (Datei)

Ein File ist ein Datenpaket im Speicher, auf einer Diskette oder einer Festplatte. Der Zugriff auf diese Daten erfolgt unter einem spezifischen Namen.

Folder (Ordner)

Im Inhaltsverzeichnis der Disketten können mit Folder bezeichnete, eigenständige Unterverzeichnisse definiert werden. Diese können Files enthalten, die in dem übergeordneten Inhaltsverzeichnis nicht mehr aufgeführt sind. Der Zugriff auf diese Files ist nur nach Öffnen des entsprechenden Folders möglich.

Formatieren

Bevor Daten auf neue magnetische Datenträger (Disketten oder Festplatten) gespeichert werden können, müssen diese formatiert, d.h. in Spuren und Sektoren eingeteilt werden.

GDOS (Graphic Device Operating System)

Das GDOS enthält die geräteunabhängigen Graphikfunktionen des GEM-VDI.

GIOS (Graphic Input/Output System)

Dieser Teil des GEM-VDI enthält den gerätespezifischen Code.

Hardcopy

Ausdruck des aktuellen Bildschirminhaltes auf dem Drucker.

Hexadezimal (Sedezimal)

Das Hexadezimalsystem beschreibt die Darstellung von Zahlen im 16er-System. Dies ist neben dem Binär- und dem Oktal-System das am meisten verwendete Zahlensystem in der Informatik.

I/O (Input/Output)

Diese Begriffe beschreiben die Daten, die von Peripheriegeräten (Massenspeicher, Tastatur, Maus usw.) in den Rechner gelesen werden oder vom Computer auf die Peripherie (Bildschirm, Drucker, Plotter usw.) ausgegeben werden.

Interface (Schnittstelle)

Ein Interface ist zum einen die elektronische Anschlußschaltung zwischen Computer und Peripheriegeräten (z.B. Centronics oder MIDI), zum anderen ein Programmteil, der die Verbindung zwischen verschiedenen, unabhängigen Programmen oder dem Bediener und Programmen ermöglicht und standardisiert (z.B. GEM zwischen Mensch und TOS).

Interrupt (Unterbrechung)

Interrupt bedeutet eine Unterbrechung des laufenden Programmes und Verzweigung in eine Maschinenroutine. Nach deren Bearbeitung wird das unterbrochene Programm wieder an derselben Stelle fortgeführt, an der es unterbrochen wurde (wie bei einem Unterprogramm). Diese Unterbrechung wird durch ein Hardware-Ereignis an einem bestimmten Eingang der CPU oder softwaremäßig durch ein Programm (z.B. durch den TRAP-Befehl) ausgelöst.

KByte (Kilobyte)

Kilobyte bedeutet eigentlich 1000 Byte, aber in der Informatik wird meist mit Zweierpotenzen gerechnet; ein KByte sind daher 210=1024 Byte.

Library (Bibliothek)

Library bedeutet in der EDV eine Sammlung von Unterprogrammen, Funktionen oder Hilfsroutinen, die in Programme mit eingebunden werden können.

Linker (Binder)

Der Linker ist ein Hilfsprogramm, welches beim Compilieren übersetzte Programme, Maschinencode-Routinen und Teile der Library zu einem lauffähigen Programm zusammenfügt.

Menü (Auswahl-Liste)

Ein Menü in einem Programm stellt mehrere Auswahlpunkte zur Verfügung, wovon eines mit Hilfe einer Zifferneingabe oder mit dem Mauszeiger ausgewählt werden kann.

Mnemo-Code

Die eigentliche Maschinensprache besteht ausschließlich aus Einsen und Nullen. Da die aus solchen Folgen aufgebauten Befehle sehr umständlich zu handhaben sind, wurden zur Vereinfachung leicht merkbare Buchstabenkombinationen eingeführt. Zum Beispiel wurde die Sequenz von Einsen und Nullen, die eine Addition zweier Zahlen bewirkt, zu dem Mnemocode ADD. In Mnemocode geschriebene Programme müssen von Übersetzern (Assemblern) in lauffähige Programme umgesetzt werden.

MIDI-Interface

Das MIDI-Interface ist eine genormte, serielle Schnittstelle zum Steuern von Musikinstrumenten durch den Computer. Es können dabei mehrere Instrumente gleichzeitig bedient werden.

Objektcode

Der Objektcode stellt den von der CPU direkt ausführbaren Programmcode dar, der nur aus Bitmustern bzw. Hexadezimalzahlen besteht. Diesen Code erhält man durch Kompilieren eines Hochsprachen-Programmes oder Assemblierung von Mnemocode.

Output

Der Output ist die Ausgabe von Daten an ein Peripheriegerät.

Parallel-Interface

siehe Centronics-Schnittstelle

Parameter

Parameter sind Variablen oder Konstanten, die an Befehle, Funktionen oder Unterprogramme zur Bearbeitung übergeben werden. Es können auch mehrere Parameter für eine Funktion benötigt werden, z.B. LINEF A, B, C, D.

PEEK

Der BASIC-Befehl PEEK liest den Inhalt einer direkt angegebenen Speicherzelle aus und übergibt ihn dem aufrufenden Programm. Dieser Befehl kann Bytes, Worte oder Langworte verarbeiten.

Peripherie

Die externen Geräte eines Computersystems wie Drucker, Diskettenlaufwerk oder Bildschirm werden Peripheriegeräte genannt.

Pixel

Ein Pixel ist das kleinste, adressierbare Graphikelement auf dem Bildschirm (Bildpunkt) oder Drucker (Matrixpunkt).

POKE

Dieser BASIC-Befehl stellt das Gegenstück zum PEEK-Befehl dar. Mit dieser Anweisung können Speicherzellen direkt verändert werden. Auch hier können Bytes, Worte und Langworte verarbeitet werden. Ein falscher POKE-Befehl kann das System abstürzen lassen, wenn damit ein Eingriff in das Betriebssystem vorgenommen wird.

Pufferspeicher (Puffer)

Ein Pufferspeicher stellt einen Speicherbereich dar, in dem Daten zwischengespeichert werden können. Mit einem solchen Puffer können Prozesse unterschiedlicher Geschwindigkeit kombiniert werden, wenn der schnellere Prozess seine Ergebnisse dort ablegt, wo der andere sie später und langsamer wieder auslesen kann (z.B. Druckerpuffer).

RAM (Random Access Memory)

Sinngemäß übersetzt bedeutet dies Speicher mit beliebigem Zugriff. Diese Speicher können gelesen und geschrieben werden. Üblicherweise bezeichnet der Begriff RAM den Halbleiterspeicher in einem Computer.

Register

Mit Register werden die prozessorinternen Speicherzellen bezeichnet, in denen nicht nur Daten gespeichert, sondern auch verknüpft werden können. Der MC 68000 besitzt insgesamt 16 Register mit 32 Bit zur freien Verfügung.

RGB (Rot-Grün-Blau)

Hierbei handelt es sich um ein Videosignal, in dem die drei Farbsignale einzeln zum Fernseher oder Monitor geführt werden, die dieser additiv mischt. Der ATARI ST kann jede dieser Grundfarben in je 8 Intensitätsstufen darstellen, so daß sich 8*8*8=512 Farbmischungen ergeben.

ROM (Read Only Memory)

ROM bezeichnet eine Speicherart, die im Gegensatz zum RAM nur ausgelesen werden kann. Die Programmierung dieser Bausteine geschieht bereits bei der Produktion. Sie werden oft zur Aufnahme des Betriebssystems eines Computers eingesetzt, da dieses dann nach dem Einschalten sofort verfügbar ist.

RS 232-Schnittstelle

Diese Schnittstelle arbeitet seriell und ist genormt. Die Datenübertragung kann in beiden Richtungen erfolgen. Die verwendeten Signalspannungen sind + und -12 Volt.

Scrollen

Scrollen bedeutet die Verschiebung eines Fensterinhaltes in eine der vier Grundrichtungen. Diese Verschiebung wird beim ATARI ST durch das Betriebssystem vorgenommen, wenn der Benutzer mit der Maus das Verschiebefeld eines Fensters anklickt.

Speicheradresse (Adresse)

Die Zellen eines Computerspeichers sind fortlaufend nummeriert, so daß durch Angabe dieser Nummer (Adresse) eine bestimmte Speicherzelle gewählt werden kann.

Steuerzeichen

Zur Auslösung von Sonderfunktionen bei Druckern oder Terminals werden entweder die nicht druckbaren ASCII-Codes von 0 bis 31 oder sogenannte Escape-Sequenzen (die Escape-Taste gefolgt von irgendeiner anderen Taste) verwendet.

TOS (Tramiel Operating System)

Das im ATARI ST verwendete Betriebssystem, das TOS, ist eine Weiterentwicklung von CP/M 68K unter Einbeziehung einiger zusätzlicher Funktionen. TOS ist dennoch mit Nichts kompatibel.

VDI (Virtual Device Interface)

Das VDI ist ein Teil des GEM, welches für die Graphikausgaben auf ein beliebiges Peripheriegerät zuständig ist.

VT52-Terminal

Dieser Menüpunkt des Desk-Menüs bewirkt eine Emulation eines Terminals, welches über die serielle Schnittstelle direkt an ein Modem oder an einen anderen Computer angeschlossen werden kann.

Wort

Ein Wort ist beim ATARI ST eine Dateneinheit, die aus 2 Byte, also 16 Bit besteht und somit einen Wert von 0 bis 65535 aufnehmen kann.

8.2 Wichtige PEEKs und POKEs

Die nun folgende Liste einiger PEEK- und POKE-Adressen stellt nicht nur eine Übersicht dar. Sie dient gleichzeitig zur Anpassung der Programme an die verschiedenen gebräuchlichen Betriebssysteme.

Der erste Wert steht für das 197 KByte lange Betriebssystem mit dem Datum 11/1985, welches von Diskette geladen wird. Diese Werte wurden in diesem Buch verwendet.

Der zweite Wert entspricht der Adresse, die für das erste verfügbare TOS gilt, welches ebenfalls geladen wird. Diese Adressen sind auch im "PEEKs & POKEs" der ersten Auflage enthalten.

Die dritte Adresse schließlich ist einzusetzen, wenn das TOS im Rechner eingebaut ist. Es gilt dabei die Version vom April 1986.

	Adresse(n)		Funktion
3086,	2256,	3086	Tastaturpuffer
3510,	2552,	3510	Zeiger auf Tastaturpuffer
3581,	2623,	3581	Maus-/Joysticktasten-Status
3582,	2624,	3582	Joystick-Stellung
3584,	2626,	3584	Uhrzeit-Puffer (6 Bytes)
3591,	2633,	3592	Joystick-Status
3652,	2694,	3652	Soundsequenz-Daten-Zeiger
9952,	8994,	9952	Mausposition horizontal
9954,	8996,	9954	Mausposition vertikal
10530,	9572,	10530	Cursorblink-Verzögerung
10531,	9573,	10531	Cursorblink-Counter
10532,	9574,	10532	Zeiger auf Fontdaten
36612,	32090,	16527388	Sounddaten Glocke
36642,	32120,	16527418	Sounddaten Klick
104536,	101474	16595294	8x16 Font Anfang

8.3 Stichwortverzeichnis

Das große BASIC-Buch zum ATARI ST ist eine ausführliche, didaktisch gut geschriebene Einführung in das BASIC des ATARI ST. Von den BASIC-Befehlen über die Problemanalyse bis zum fertigen Algorithmus lernt man schnell und sicher das Programmieren. Übungsaufgaben helfen, das Gelernte zu vertiefen. Gleichzeitig erhält der BASIC-Programmierer ein praxisbezogenes Nachschlagewerk.



Aus dem Inhalt:

- Datenfluß- und Programmablaufpläne
- fortgeschrittene Programmiertechniken
- Grafikprogrammierung
- Mehrdimensionale Felder
- Sortierverfahren
- Dateiverwaltung
- BASIC unter GEM
  und vieles mehr

Kampow
Das große BASIC-Buch zum ATARI ST
268 Seiten, DM 39,–
ISBN 3-89011-121-1

Ein Buch für jeden, der das Betriebssystem der Zukunft verstehen und anwenden und die gigantische GEM-Bibliothek nutzen will! Von grundlegenden Informationen wie der Organisation des GEM im ATARI ST über die verwendbaren Programmiersprachen bis zu den Funktionen des Virtual Device Interface und des Application Environment System ist alles sehr gründlich und exakt erklärt!



Aus dem Inhalt:

- Die Grundstrukturen der GEM-Komponenten VDI und AES
- Die Wahl der richtigen Programmiersprache
- Einführung in C und Assembler
- Beschreibung und Benutzung des Entwicklungspaketes
- Der Editor
- Der C-Compiler
- Der Assembler
- Der Linker
- Aufbau, Funktion und Programmierung des VDI und AES
- Beispielprogramme in C und Assembler

Szczepanowski, Günther
Das große GEM-Buch zum ATARI ST
459 Seiten, DM 49,–
ISBN 3-89011-125-4

DAS STEHT DRIN:

Schlagen Sie dem Betriebssystem Ihres ATARI ST ein Schnippchen. Wie? Mit PEEKS & POKES natürlich! Dieses Buch erklärt leichtverständlich den Umgang mit einer riesigen Anzahl wichtiger POKES und ihren Anwendungsmöglichkeiten. Nebenbei wird der interne Aufbau Ihres neues ATARI ST prima erklärt.

Aus dem Inhalt:

- Ein-Blick in den ATARI ST
- Innere Konfiguration und Schnittstellen
- Die intelligente Tastatur
- Die Maus als Malstift
- Die internen Speicher
- Zeiger und Stacks
- Diskettenhandling
- Computer-Einmaleins
- Das TOS
- GEM
- Interpreter/Compiler
- Programmiersprachen
- Ein/Ausgaben

UND GESCHRIEBEN HAT DIESES BUCH:

Stefan Dittrich ist Informatik-Student und ausgezeichneter Kenner des ATARI ST. Er besaß als einer der ersten das ATARI-Entwicklungspaket und arbeitet seit der Zeit zusammen mit dem Hause DATA BECKER an neuer Software und interessanten Hardwaretricks.

ISBN 3-89011-148-3